Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

На правах рукописи

Барков Вячеслав Валерьевич

КЛАССИФИКАЦИЯ ПРОТИВОПРАВНЫХ И НЕЖЕЛАТЕЛЬНЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ В ПОТОКОВОМ РЕЖИМЕ

Специальность 2.3.6. Методы и системы защиты информации, информационная безопасность

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель
Заслуженный деятель науки РФ,
доктор технических наук,
профессор Шелухин Олег Иванович

Москва – 2024

СОДЕРЖАНИЕ

Введение 5

Глава 1 Анализ основных подходов к классификации противоправных, вредоносных и нежелательных мобильных приложений на основе анализа сетевого трафика методами машинного обучения в условиях априорной неопределенности и изменения характеристик трафика 13

1.1 Ограничение доступа к программным приложениям в соответствии с законодательством Российской Федерации 13

1.2 Вредоносные Android-приложения 17

1.3 Решение задачи пакетной классификации классическими методами машинного обучения 22

1.4 Применение методов глубокого обучения 23

1.5 Потоковая классификация 26

1.6 Смена концепта 29

1.6.1 Контролируемые методы обнаружения смены концепта 29

1.6.2 Полуконтролируемые методы обнаружения смены концепта 31

1.6.3 Неконтролируемые методы обнаружения смены концепта 31

1.6.4 Методы обнаружения смены концепта на основе глубокого обучения 34

Выводы 35

Глава 2 Классификация противоправных, вредоносных и нежелательных мобильных приложений в режиме offline в условиях наличия фонового трафика 36

2.1 Постановка задачи 36

2.2 Сбор данных 41

2.3 Вычисление атрибутов 43

2.4 Выбор алгоритмов и метрик оценки качества классификации 47

2.5 Методика отбора атрибутов 48

2.6 Выбор атрибутов классификации 49

2.7 Результаты классификации мобильных приложений для выбранных наборов атрибутов 51

2.8 Влияние наличия фонового трафика на качество классификации 51

2.9 Классификация мобильных приложений на основе анализа сетевого трафика при наличии фонового трафика и класса «Неизвестное приложение». 56

2.10 Классификация мобильных приложений на основе анализа сетевого трафика при наличие фонового трафика с помощью автокодировщиков 61

2.10.1 Эффективность автокодировщиков 61

2.10.2 Модель классификации с использованием АК 63

2.10.3 Результаты классификации с помощью АК 65

Выводы 70

Глава 3 Разработка модели обнаружения смены концепта на основе автокодировщиков 72

3.1 Обнаружение смены концепта с помощью автокодировщиков 72

3.1.1 Обучение модели обнаружения смены концепта 72

3.1.2 Обнаружение смены концепта в потоке 73

3.1.3 Восстановление атрибутов приложений 74

3.1.4 Алгоритм обнаружения смены концепта 77

3.2 Обнаружение смены концепта с учётом эффекта старения данных 84

Выводы 86

Глава 4 Исследование потоковой классификации мобильных приложений на основе анализа сетевого трафика 87

4.1 Классификация мобильных приложений на основе анализа сетевого трафика в потоковом режиме 87

4.2 Разработка программного комплекса «Система анализа трафика» 90

4.2.1 Анализ предметной области 90

4.2.2 Инфологическое проектирование 93

4.2.3 Проектирование базы данных: даталогическое и физическое проектирование 97

4.2.4 Проектирование и разработка серверного программного обеспечения..... 98

4.2.5 Проектирование и разработка мобильного приложения для сбора сетевого трафика 101

Выводы 104

Заключение 106

Список литературы 108

Приложение А Результаты классификации противоправных, нежелательных и вредоносных мобильных приложений на основе анализа сетевого трафика при наличии и отсутсвии шифрования с применением алгоритмов KNN и Random Forest 121

Приложение Б Программная реализация модели обнаружения смены концепта на основе автокодировщика 122

Приложение В Свидетельство о государственной регистрации программы для ЭВМ 131

Приложение Г Акт об использовании в учебном процессе ФГБОУ ВО МТУСИ научных результатов диссертационной работы 132

Приложение Д Акт об использовании результатов диссертационной работы в АО «Лаборатория Касперского»..... 133

ВВЕДЕНИЕ

Актуальность темы исследования. Задача выявления мобильных приложений, осуществляющих распространение противоправного, нежелательного или ***вредоносного контента***, приобретает особую актуальность в связи с активным развитием мобильных устройств. Под противоправным контентом понимается информация, содержание которой противоречит законодательству Российской Федерации. В числе типовых нарушений, которые должны выявляться в противоправном контенте — призывы к массовым беспорядкам, оскорбление общества, государственной власти, официальных государственных символов, конституции или исполнительной власти, призывы к суицидам, информация об изготовлении и приобретении наркотиков и др.

Федеральное законодательство предусматривает порядок ограничения к таким ресурсам в сети Интернет и к соответствующим программным приложениям. Ограничение доступа к приложениям предусматривается в случаях распространения информации с нарушением авторских и/или смежных прав, а также в случае установления факта неисполнения организатором распространения информации в сети Интернет обязанностей, предусмотренных законодательством. Для операторов мобильной связи информация об использовании пользователями тех или иных приложений необходима для получения статистики по наиболее востребованным, в том числе противоправным. При необходимости мониторинг приложений в потоковом режиме обеспечивает ограничение доступа к подобным сетевым ресурсам.

Для решения подобных задач широкое распространение получили методы интеллектуального анализа данных (Data Mining, DM) и машинного обучения (Machine Learning, ML), позволяющие адаптироваться к непрерывно изменяющейся структуре Интернет-ресурсов и учитывающие специфику сетевого трафика. Внедрение таких методов позволяет с достаточно высокой эффективностью производить классификацию, анализ и фильтрацию сетевого

трафика мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента.

Вместе с тем в известных работах, посвященные проблеме классификации приложений на основе анализа сетевого трафика, слабо учитывается требование выявления неизвестного сетевого трафика. При проектировании моделей классификации приложений он полностью исключается в предположении наличия только известных классов, обучение осуществляется на данных из ограниченного числа классов приложений и тестируется с помощью других данных из тех же известных классов.

Отсутствие полной и достоверной информации о структуре фонового трафика значительно снижает качество классификации интересуемых мобильных приложений. Известным методом повышения качества классификации является использование архитектуры искусственных нейронных сетей (ИНС) автокодировщик (АК).

Степень разработанности темы. Теоретическую базу диссертации в области методов ML в области информационной безопасности составляют работы таких ученых, как *Зегжда П.Д., Зегждаа Д.П., Лаврова Д.С., Козачок А.В., Марков А.С., Молдовян Н.А., Крундышева В.М., Калинин М.О.*, Котенко И.В., Шелухин О.И., M. Pietrzyk, Z. Chen, B. Yang, J. Erman, K. Balachandran, J.H. Broberg, T. Bujlow, V. Carela-Español, C.C. Aggarwal, Y. Wang, G.S.o Han, J. Erman, M. Arlitt, A. Mahanti и др. Однако задаче классификации мобильных приложений на основе анализа сетевого трафика было уделено недостаточно внимания.

Вышесказанное обусловливает актуальность настоящего исследования, направленного на повышение эффективности классификации мобильных приложений на основе анализа сетевого трафика методами машинного обучения в потоковом режиме.

Целью диссертационного исследования является повышение эффективности классификации мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента, на

основе анализа сетевого трафика методами машинного обучения в потоковом режиме.

Достижение поставленной цели предусматривает решение **частных задач:**

1) Сравнительный анализ известных алгоритмов классификации мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента, на основе анализа сетевого трафика в условиях априорной неопределенности в режиме offline.
2) Разработка нового алгоритма на основе использования ИНС в виде автокодировщика.
3) Разработка модели обнаружения смены концепта в наблюдаемых атрибутах при классификации мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента, на основе анализа сетевого трафика.
4) Разработка алгоритмов классификации мобильных приложений на основе анализа сетевого трафика в потоковом режиме с «конечной» и «бесконечной» памятью на базе созданных репрезентативных выборок;
5) Разработка программного комплекса (ПК) «Система анализа трафика» (САТ) для автоматизации процесса классификации мобильных приложений с помощью анализа сетевого трафика.

Объект исследования: сетевой трафик, генерируемый мобильными приложениями.

Предмет исследования: методы ML для классификация мобильных приложений на основе анализа сетевого трафика в условиях априорной неопределенности числа приложений.

Методы исследования: методы математического моделирования, теория вероятности и математической статистики, методы машинного обучения и интеллектуального анализа (обработки) данных. Методологической основой исследования является системный подход.

Основные положения, выносимые на защиту:

1) Методика отбора рационального числа атрибутов на основе анализа их информативности при фиксировании допустимой вероятности ложной классификации. При этом определены ограничения на структуру анализируемых данных по числу пакетов и потоков, в то время как ***общепринятые*** подходы предполагают расчет характеристик на основе данных всего потока.
2) Модифицированный алгоритм классификации мобильных приложений в условиях неконтролируемого фонового трафика, ***отличающийся от известных*** алгоритмов каскадным включением нейронной сети с архитектурой АК, выполняющей предварительную фильтрацию, и вторичной модели классификации.
3) Статистическая ***модель обнаружения смены концепта*** при классификации мобильных приложений на основе анализа сетевого трафика, ***отличающаяся от известных включением АК*** в качестве базовой модели обнаружения смены концепта, в котором момент наступления смены концепта определяется посредством оценок ошибок восстановления анализируемых приложений и превышения пороговых значений.
4) ***Новый алгоритм обнаружения смены концепта*** мобильных приложений в потоковом режиме с обработкой в скользящем окне в режиме накопления с «конечной памятью», как с равномерной, так и неравномерной интенсивностью поступления данных, ***отличающийся от известных учетом «старения» данных*** в окне обработки и негауссовским характером изменяющихся параметров классифицируемых приложений.
5) Модифицированный алгоритм Adaptive Random Forest (MARF) со встроенной моделью обнаружения смены концепта, позволяющей обнаруживать смену концепта не только во время обучения, но и во время предсказания, т.к. не использует истинные метки, осуществляет

классификацию быстрее чем алгоритмы Random Forest (RF), Hoeffding Adaptive Tree (HAT), K nearest neighbors (KNN), Oza Bagging (OB).

Теоретическая значимость исследования состоит в разработке и совершенствовании математических моделей и алгоритмов, позволяющих путём применения методов машинного обучения осуществлять в потоковом режиме классификацию мобильных приложений, осуществляющих распространение противоправного, вредоносного и нежелательного контента, в условиях априорной неопределенности относительно состава и числа классифицируемых приложений и возможной смены концепта.

Научная новизна состоит в следующем:

1) Методика отбора значимых атрибутов классификации, обеспечивающая повышение качества классификации, высокую достоверность классификации мобильных приложений, осуществляющих шифрование сетевого трафика, более 90% при ограниченном размере обучающей выборки (300 потоков с 16-58 пакетами в каждом, в зависимости от приложения), в то время как ***общепринятые подходы*** предполагают расчет характеристик по данным всего потока. Предложенная методика является инвариантной по отношению к разным типам сетевого трафика.
2) ***Алгоритм классификации*** мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента, состоящий из последовательно включенных АК и типовой модели классификации, ***обеспечивает в условиях*** априорной неопределенности и неконтролируемого ***фонового трафика повышение достоверности*** (accuracy) ***классификации приложений на 7% по сравнению с известными алгоритмами***, ***не требуя разметки фоновых приложений*** в случае их внезапного появления.
3) ***Модель обнаружения смены концепта*** классифицируемых мобильных приложений, ***отличающаяся от известных включением АК*** (для каждого приложения), в которой момент смены концепта определяется по ошибкам восстановления анализируемых мобильных приложений и

превышению пороговых значений, ***что повышает точность обнаружения смены концепта*** в потоковом режиме.

4) ***Алгоритм обнаружения смены концепта*** и классификации мобильных приложений, осуществляющих распространение противоправного, нежелательного или вредоносного контента в потоковом режиме с накоплением и обработкой в скользящем окне в условиях ограниченной памяти для равномерной и неравномерной интенсивности поступления данных, ***отличающийся от известных алгоритмов учетом «старения» данных***.
5) ***Модифицированный адаптивный MARF в отличие от стандартного алгоритма ARF***, использующего модель обнаружения смены концепта только на этапе обучения и использующего истинные метки класса, ***позволяет обнаруживать смену концепта на этапе предсказания***.

Практическая ценность работы заключается в разработке алгоритмов и реализации программного комплекса для классификации мобильных приложений, осуществляющих распространение противоправного, вредоносного и нежелательного контента, на основе анализа сетевого трафика в потоковом режиме в условиях априорной неопределенности состава и числа классифицируемых приложений, в условиях смены концепта.

Сформированная экспериментальная база данных сетевого трафика мобильных приложений, которая может быть использована в системах обнаружения вторжений, для блокировки мобильных приложений, осуществляющих распространение противоправного, вредоносного и нежелательного контента, в том числе приложений, использующих шифрование сетевого трафика.

Достоверность результатов диссертационной работы подтверждается сходимостью результатов имитационного моделирования с результатами экспериментальных данных, корректным использованием современного математического аппарата, а также достаточно широким рядом публикаций,

обсуждением основных положений со специалистами на научных конференциях и семинарах.

Внедрение результатов работы.

Результаты диссертационных исследований, подтвержденные соответствующими актами внедрения, используются в АО «Лаборатория Касперского» при разработке межсетевых экранов, а также в учебном процессе МТУСИ.

Апробация результатов.

Основные результаты работы обсуждались и получили одобрение на конференциях: Международная научно-техническая конференция «Телекоммуникационные и вычислительные системы – 2018».; Международная конференция «Technology & entrepreneurship in digital society» ; Научно-техническая конференция РОСИНФОКОМ-2018 «Беспроводная связь и информационная безопасность интернета»; Международная научно-техническая конференция «Фундаментальные проблемы радиоэлектронного приборостроения «INTERMATIC-2018»; IX Всероссийская научно-техническая конференция «Безопасные информационные технологи»; Международная научно-техническая конференция «Systems of signals generating and processing in the field of on board communications - 2019»; II Всероссийская научная школа-семинар «Современные тенденции развития методов и технологии защиты информации».

Публикации.

Основные положения диссертации опубликованы в 18 научных печатных работах, в том числе: 5 – в научных журналах перечня ВАК; 1 – в научных рецензируемых изданиях по базе Scopus; 11 – в материалах конференций и других изданиях. Получено свидетельство о Государственной регистрации программы для ЭВМ.

Соответствие паспорту специальности. Диссертация соответствует п.13. «Методы и модели выявления и противодействия распространению ложной и вредоносной информации» и п.15. «Принципы и решения (технические, математические, организационные и др.) по созданию новых и совершенствованию

существующих средств защиты информации и обеспечения информационной безопасности» паспорта специальности 2.3.6.

Личный вклад автора. Основные научные результаты, в том числе разработка алгоритмов обнаружения смены концепта, методических рекомендаций по подбору параметров алгоритмов, получены автором лично. Вклад соавторов ограничивался постановкой задач на исследования и обсуждением полученных результатов.

Связь работы с научными программами, темами, грантами. Исследования выполнялись в инициативном порядке, в рамках работы по гранту аспирантам, соискателям и молодым ученым на исследования, направленные на обеспечение информационной безопасности для задач цифровой экономики и при государственной поддержке ведущих научных школ Российской Федерации в области информационной безопасности.

ГЛАВА 1 АНАЛИЗ ОСНОВНЫХ ПОДХОДОВ К КЛАССИФИКАЦИИ ПРОТИВОПРАВНЫХ, ВРЕДОНОСНЫХ И НЕЖЕЛАТЕЛЬНЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ АНАЛИЗА СЕТЕВОГО ТРАФИКА МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ В УСЛОВИЯХ АПРИОРНОЙ НЕОПРЕДЕЛЕННОСТИ И ИЗМЕНЕНИЯ ХАРАКТЕРИСТИК ТРАФИКА

1.1 Ограничение доступа к программным приложениям в соответствии с законодательством Российской Федерации

В целях ограничения доступа к сайтам в сети Интернет, содержащим информацию, распространение которой в Российской Федерации запрещено, была создана единая автоматизированная информационная система "Единый реестр доменных имен, указателей страниц сайтов в сети Интернет и сетевых адресов, позволяющих идентифицировать сайты в сети Интернет, содержащие информацию, распространение которой в Российской Федерации запрещено", называемая далее реестром.

В реестр включается доменные имена, указатели страниц сайтов в сети Интернет, содержащих информацию, распространение которой запрещено в Российской Федерации, а также сетевые адреса, позволяющие идентифицировать сайты в сети Интернет, содержащие такую информацию.

Основаниями для включения в реестр являются решения уполномоченных Правительством Российской Федерации федеральных органов исполнительной власти в отношении:

− материалов с порнографическими изображениями несовершеннолетних, объявлений о привлечении несовершеннолетних в качестве исполнителей для участия в зрелищных мероприятиях порнографического характера;

− информации о способах, методах разработки, изготовления и использования наркотических средств, психотропных веществ и их прекурсоров, новых потенциально опасных психоактивных веществ, местах их приобретения, способах и местах культивирования наркосодержащих растений;

− информации о способах совершения самоубийства, а также призывов к совершению самоубийства;

− информации о несовершеннолетнем, пострадавшем в результате противоправных действий (бездействия), распространение которой запрещено федеральными законами;

− информации, нарушающей требования федеральных законов [2, 3] о запрете деятельности по организации и проведению азартных игр и лотерей с использованием сети "Интернет" и иных средств связи, а также информации, обеспечивающей возможность совершения действий по переводу денежных средств через иностранных поставщиков платежных услуг, включенных в перечни, предусмотренные федеральными законами;

− информации, содержащей предложения о розничной продаже дистанционным способом алкогольной продукции, спиртосодержащей пищевой продукции, этилового спирта, спиртосодержащей непищевой продукции, розничная продажа которой ограничена или запрещена законодательством о государственном регулировании производства и оборота этилового спирта, алкогольной и спиртосодержащей продукции и об ограничении потребления (распития) алкогольной продукции;

− информации, направленной на склонение или иное вовлечение несовершеннолетних в совершение противоправных действий, представляющих угрозу для их жизни, здоровья либо для жизни, здоровья иных лиц;

− информации, содержащей предложение о розничной торговле лекарственными препаратами, в том числе дистанционным способом, розничная торговля которыми ограничена или запрещена в соответствии с законодательством об обращении лекарственных средств, информации, содержащей предложение о розничной торговле лекарственными препаратами, в том числе дистанционным способом, лицами, не имеющими лицензии и разрешения на осуществление такой деятельности, если получение лицензии и разрешения предусмотрено законодательством об обращении лекарственных средств;

– информации, содержащей сведения о лицах, в отношении которых в соответствии с федеральными законами [4, 5] обеспечивается конфиденциальность;

– информации о способах, методах самодельного изготовления взрывчатых веществ и взрывных устройств, незаконного изготовления или переделки оружия, основных частей огнестрельного оружия, а равно незаконного изготовления боеприпасов, за исключением информации о способах, методах самостоятельного снаряжения патронов к гражданскому огнестрельному длинноствольному оружию;

– информации, пропагандирующей нетрадиционные сексуальные отношения, предпочтения, педофилию, смену пола.

Основанием включения в реестр является также решение суда о признании информации запрещённой к распространению в Российской Федерации или постановление судебного пристава-исполнителя об ограничении доступа к информации, порочащей честь, достоинство или деловую репутацию гражданина либо юридического лица.

Федеральный закон [1] предусматривает не только порядок ограничения к сайтам, но и ограничения к программным приложениям.

Ограничение доступа к программным приложениям предусматривается в случаях распространения с помощью программных приложений информации с нарушением авторских и (или) смежных прав, а также в случае установления факта неисполнения организатором распространения информации в сети Интернет обязанностей, предусмотренных федеральным законом [1].

В случае нарушением авторских и (или) смежных прав правообладатель вправе обратиться в федеральный орган исполнительной власти с заявлением о принятии мер по ограничению доступа к информационным ресурсам, в том числе программным приложениям, на основании вступившего в силу судебного акта. Федеральный орган исполнительной власти в течение трёх рабочих дней должен определить владельца программного приложения, а также владельца информационного ресурса, на котором размещено программное приложение,

посредством которого обеспечивается доступ к объектам авторских и (или) смежных прав или к информации, необходимой для их получения без разрешения правообладателя. Далее федеральный орган исполнительной власти должен направить владельцу информационного ресурса, на котором размещено программное приложение, уведомление на русском и английском языках о нарушении исключительных прав на объекты авторских и (или) смежных прав с указанием наименования произведения, его автора, правообладателя и наименования программного приложения или иной информации, позволяющей идентифицировать программное приложение, посредством которого осуществляется доступ к объектам авторских и (или) смежных прав. Федеральный орган исполнительной власти должен зафиксировать дату и время направления уведомления. В течение одного рабочего дня с момента получения уведомления владелец информационного ресурса, на котором размещено программного приложение, обязан проинформировать об этом владельца приложения и уведомить его о необходимости незамедлительного ограничения доступа к объектам авторских и (или) смежных прав, распространяемых без разрешения правообладателя. В течение одного рабочего дня с момента получения уведомления от владельца информационного ресурса, на котором расположено программное приложение, владелец программного приложения обязан ограничить доступ к объектам авторских и (или) смежных прав. В случае отказа или бездействия владельца программного приложения владелец информационного ресурса, на котором размещено программное приложение, обязан ограничить доступ к соответствующему программному приложению не позднее истечения трёх рабочих дней с момента получения уведомления. В случае непринятия владельцем информационного ресурса, на котором размещено программное приложение, операторам связи направляется информация, необходимая для принятия мер по ограничению доступа к программному приложению. Отмена ограничений осуществляется федеральным органом исполнительной власти на основании вступившего в силу судебного акта об отмене ограничения доступа к программному приложению в течение трёх рабочих дней.

Аналогичный порядок ограничения доступа и отмены ограничения устанавливается в случае установления факта неисполнения организатором распространения информации в сети Интернет обязанностей, предусмотренных федеральным законом [1].

1.2 Вредоносные Android-приложения

До недавнего времени большинство задач, связанные со взломом и проведением атак были выполнимы только с использованием персональных компьютеров. С развитием технологий мобильные устройства превратились в полноценные вычислительные машины и могут выполнять те же функции, что и персональный компьютер [6]. Обычно для проведения атак злоумышленники использовали операционные системы на основе Linux. Android также является операционной системой на основе Linux. Это привело к созданию множеству приложений для осуществления атак, и теперь нет необходимости в использовании специальных операционных систем и персональных компьютеров для проведения атак. Однако такие приложения часто требуют прав суперпользователя, которые по умолчанию не предоставляются. Некоторые приложения доступны для загрузки в Google Play, а некоторые на других внешних ресурсах.

Примерами таких приложений являются Orbot [7], Fing [8], DriveDroid [9], PixelKnot [10], WiFi WPS WPA TESTER [11], Change My MAC – Spoof WiFi MAC [12], Hackode, SSHDroid, SSLStrip, HashDecrypt, Whatscan, NetCut, WPS Connect, Market Helper [13], Androrat [14], Zanti [15], Nmap for Android [16], Droid Pentest [17], USB Clever [18], AppUse [19], Kali Linux NetHunter [20], Intercepter-N [21], WiFi Kill, APKInspector, dSploit, AnDOSid, Penetrate Pro, Faceniff, Shark, Whatsapp sniffer, WIBR Plus, Lucky Patcher, и др.

Первые 6 приложений доступны в официальном магазине приложений для Android Google Play.

Приложение Orbot обеспечивает анонимность и конфиденциальность в ети Интернет. Оно позволяет скрыть реальный IP-адрес при помощи распределённой сети ретранслирующих сетевых устройств, запущенных волонтёрами по всему

миру. Приложение способно направить весь TCP-трафик через Tor, но для этого необходимы права суперпользователя.

Приложение Fing является быстрым сетевым сканером, который можно использовать для обнаружения устройств, подключённых к сети Wi-Fi, отображения местоположения этих устройств, обнаружения нарушителей, оценивания угроз сетевой безопасности, устранения проблем с сетью, а также достижения лучшей производительности сети. В руках злоумышленников это приложение является Wi-Fi сканером, сканером портов, инструментом отслеживания служб, и инструментом для работы с протоколами DNS и ICMP.

Приложение DriveDroid предназначено для загрузки персонального компьютера из образов операционной системы, хранящихся на мобильном устройстве. Приложение включает удобное меню загрузки, позволяющее выбрать операционную систему для загрузки. В настоящее время доступно около 35 различных систем.

Приложение PixelKnot предназначено для обмена сообщениями, использующее технологии стеганографии, для скрытия сообщений.

Приложение WIFI WPS WPA TESTER предназначено для проверки корректной настройки протокола WPS точки доступа Wi-Fi.

Приложение Change My MAC – Spoof Wifi MAC предназначено для подмены MAC-адреса устройства. Может использоваться для получения неограниченного по времени доступа к точке доступа Wi-Fi в аэропортах.

Приложение Hackode является инструментарием злоумышленника, инструментом пентестеров, IT администраторов и специалистов в области кибербезопасности, позволяющее выполнять различного рода задачи, например, сканирование выполняемых эксплойтов.

Приложение SSHDroid является реализацией SSH-сервера для Android, что позволяет подключаться к устройству через персональный компьютер и выполнять команды или редактировать файлы.

Приложение SSLStrip для Android является инструментом, который явно захватывает HTTP-трафик в сети, следит за HTTPS ссылками и переадресациями, а

затем преобразует ссылки в аналогичные HTTP ссылки или в похожие омографические HTTPS ссылки.

Приложение HashDecrypt позволяет восстанавливать захешированное значение, используя атаку «Перебор по словарю». Приложение поддерживает 10 алгоритмов хеширования, MD2, MD4, MD5, SHA1, SHA-256, SHA-384, SHA-512, Tiger, RIPEMD-128, RIPEMD-160.

Приложение Whatscan разработано для взлома учетных записей WhatsApp, не требует прав суперпользовател, так как использует подлинный метод для доступа к WhatsApp.

Приложение NetCut предназначено для быстрого обнаружения пользователей Wi-Fi. Приложение может обнаруживать пользователей, не имея действительного IP-адреса. Также возможно подключение и отключение любых пользователей сети.

Приложение WPS Connect предназначено для проверки уязвимости точки доступа Wi-Fi с включённым протоколом WPS. Приложение используются злоумышленниками для взлома Wi-Fi сетей, с включённым протоколом WPS.

Следующие приложения не доступны в Google Play, но их можно загрузить со сторонних источников.

Приложение Market Helper является инструментом для Android, который помогает подделывать пользователям на любое другое устройство. Для этого требуются права суперпользователя. Данное приложение позволяет тем самым устанавливать несовместимое или запрещенное в стране приложения.

Приложение Androrat – удалённый инструмент администрирования Android. Приложение может выступать в качестве клиента или сервера. Приложение позволяет получить с целевого устройства контакты, перевести устройство в режим виброзвонка, получать журналы вызовов, получать и отправлять сообщения, узнавать местоположения устройства, транслировать видео, отслеживать состояние телефона в реальном времени использовать камеру и многое другое.

Приложение zANTI является мобильным набором инструментов для тестирования на проникновения, который позволяет специалистам в сфере

безопасности оценивать уровень угрозы для сети, имитировать опытного злоумышленника для выявления вредоносных техник, используемые для взлома корпоративных сетей.

Приложение Nmap for Android – сканер безопасности с открытым исходным кодом, используемый для исследования сети. Он работает не зависимо от наличия прав суперпользователя.

Приложение Droid Pentest является платформой для проведения тестирования на проникновение.

Приложение USB Cleaver предназначено для тихого получения информации с целевой машины под управлением ОС Windows, такой как хеши паролей, секреты LSA, IP информацию.

Приложение AppUse является виртуальной машиной. Это уникальная платформа для тестирования безопасности мобильного приложения, включающая в себя специализированные инструменты. Ядро AppUse является специальным враждебным Android ROM, специально разработанным для тестирования безопасности приложения и содержащим изменённую среду выполнения, которая работает поверх настроенного эмулятора. AppUse включает в себя все, что может понадобиться пентестеру для того, чтобы запустить и проверить целевое приложение: эмулятор Android, инструменты разработчика, необходимые SDK, декомпиляторы, дизамблеры и т.д.

Приложение Kali Linux NetHunter является мощной платформой для тестирования на проникновения. Приложение обладает всей мощью Kali.

Приложение Intercepter-N является Android-инструментом для злоумышленников. Оно позволяет перехватывать и анализировать незашифрованный трафик через Wi-Fi сеть.

Приложение WiFi Kill используется для того, чтобы блокировать Wi-Fi соединения для других пользователей.

Приложение APKInspector предназначено для анализа и визуализации файлов APK, в том числе программного кода, хранящегося в файлах DEX. Используется для анализа вредоносных приложений.

Приложение dSploit является набором инструментов для анализа сети и проведения тестирования на проникновения. Основной задачей приложения является предложение специалистам в сфере IT безопасности набора инструментов для проведения оценки сетевой безопасности с помощью мобильных устройств. Приложение позволяет отслеживать сеть, идентифицировать действующие хосты операционных систем и запущенные службы, искать известные уязвимости, взламывать процедуры входа TCP-протоколов, проводить атаки типа человек посередине для получения пароля, управлять сетевым трафиком в режиме реального времени.

Приложение AnDOSid используется для запуска DoS атак с мобильного устройства. Приложение было разработано как инструмент для стрессового тестирования. Может использоваться злоумышленниками для вывода из строй Web-серверов.

Приложение Penetrate Pro используется для подсчёта ключей WPA/WEP для некоторых маршрутизаторов, то есть может быть использован злоумышленниками для расшифровки Wi-Fi-трафика.

Приложение Faceniff является приложением поиска и прерывания профилей Web-сессий через Wi-Fi.

Приложение Shark предназначено для отслеживания действий других устройств. Оно было разработано для офисного персонала, чтобы иметь возможность отслеживать действия членов команды или сотрудников офиса.

Приложение Whatsapp sniffer предназначено для кражи переписок, видео или аудио в мессенджере WhatsApp.

Приложение WIBR Plus предназначено для тестирования безопасности WPA/WPA 2 PSK Wi-Fi сетей. Приложение фиксирует беспроводное проникновение в сеть.

Приложение Lucky Patcher является инструментом для Android, позволяющим удалять рекламу, изменять права доступа приложений, создавать резервные копии и восстанавливать приложения, обходить проверку наличия

лицензии на специальные приложения и многое другое. Для некоторого функционала требуются права суперпользователя.

1.3 Решение задачи пакетной классификации классическими методами машинного обучения

Пакетная классификация сконцентрирована на получении лучшей оценки на ограниченном наборе данных, когда число доступных экземпляров порядка сотен. В идеальном случае все данные, которые доступны, должны быть использованы для обучения модели классификации, но в таком случае не останется данных для тестирования модели.

Метод отложенной выборки (holdout) разделяет доступные данные на два непересекающихся множества. Одно из множеств используется для обучения, называемое обучающим множеством, обучающим набором или обучающей выборкой, а второе, называемое тестовым множеством, отложенным множеством, – для тестирования. Разделение этих множество гарантирует измерение эффективности обобщения. Наиболее частые пропорции, используемые на практике, 50% на обучение и 50% на тестирование и 2/3 на обучение и 1/3 на тестирование. Главным недостатком этого метода является то, что данные не используются эффективно, т.к. не все данные используются для обучения. Оценка точности, полученная на одной выборке, может значительно зависеть от того, каким образом разделены данные. Для устранения этого эффекта процесс случайной выборки повторяется несколько раз. В результате получаются различные разделения всех доступных данных на обучающую и тестовую выборку. Оценка точности усредняется. Однако это нарушает предположение о том, что обучающая и тестовая выборка независимы.

В отличие от метода отложенной выборки, метод перекрёстной проверки (cross-validation) максимизирует использование экземпляров для обучения и тестирования. В k-кратной перекрёстной проверке (k-fold cross-validation) данные случайно разделяются на k независимых и приблизительно равных по размеру множеств. Процесс оценки повторяется k раз, каждый раз использую разное

множество в качестве отложенной выборки, в то время как остальные множества объединяются и используются в качестве обучающей выборки. Окончательная оценка точности получается путем деления общего количества правильных классификаций на общее количество экземпляров. В этой процедуре каждый доступный экземпляр используется $k-1$ раз для обучения и ровно один раз для тестирования. Этот метод по-прежнему подвержен несбалансированному распределению классов между множествами. Пытаясь решить эту проблему, стратифицированная перекрестная проверка равномерно распределяет метки по множествам, чтобы приблизительно отразить распределение меток во всех данных. Повторная перекрестная проверка повторяет процедуру перекрестной проверки несколько раз, каждый раз с различным случайным разделением множеств, что позволяет измерить дисперсию оценки точности.

Рассмотрев различные проблемы с оценкой производительности в пакетном режиме, сообщество машинного обучения остановилось на стратифицированной десятикратной перекрестной проверке в качестве стандартной процедуры оценки. Для повышения надежности обычно используются десять повторений десятикратной перекрестной проверки.

Известными алгоритмами многоклассовой классификации в задачах классификации сетевого трафика [22, 23] являются логистическая регрессия (Logistic Regression), наивный байесовский классификатор (Naive Bayes), K ближайших соседей (KNN), алгоритмы на основе деревьев решений (ID3 [24], C4.5 [25], CART [26], CHAID [27], QUEST [28]). При решении сложных задач классификации используется композиция алгоритмов: бэггинг [29], бустинг (AdaBoost [30], Gradient Boosting [31], CatBoost [32], LightGBM [33], XGBoost [34]), стэккинг и случаный лес (RandomForests [35], IsolationForest [36]).

1.4 Применение методов глубокого обучения

Автокодировщики – это модели глубокого обучения, которые обладают способностью изучать закодированное представление на нижнем слое, а затем воспроизводить входные данные на выходном уровне.

Автокодировщик это ИНС, которая сначала кодирует входной сигнал в некоторое скрытое состояние, размерность которого, как правило, меньше размерности входного сигнала, а затем, из скрытого состояния снова разворачивает (декодирует) данные в другое, новое состояние. ИНС состоят из L слоев нейронов, где каждый i-й слой $l^{(i)}$ последовательно соединен через синапсы со связанными весами $W^{(i)}$. Простой АК состоит из входного слоя, одного или нескольких скрытых слоев и выходного слоя того же размера, что и входной слой (рис. 1.1).

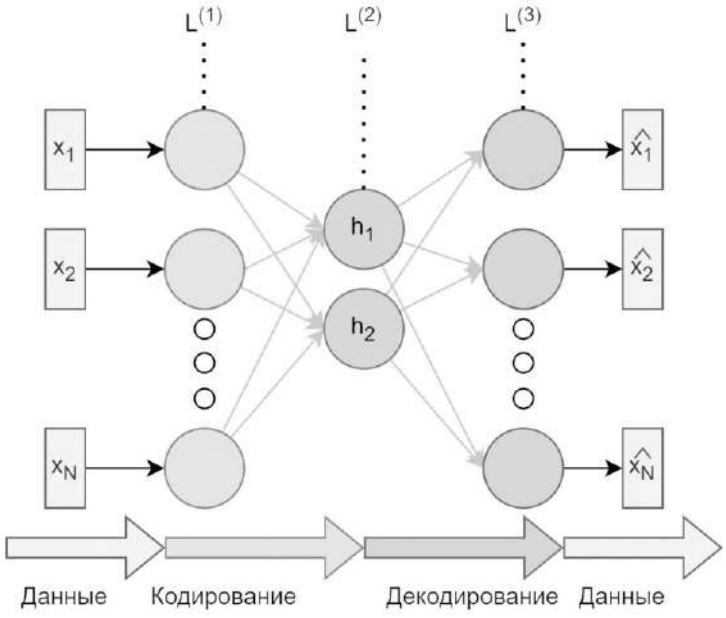


Рисунок 1.1 – Структура автокодировщика

Пусть $l^{(i)}$ обозначает i-й слой в ИНС, а $\left\|l^{(i)}\right\|$ обозначает количество нейронов в этом слое $l^{(i)}$. Веса, соединяющие $l^{(i)}$ и $l^{(i+1)}$ обозначаются как матрица $W^{(i)}$ размера $\left\|l^{(i)}\right\|$ на $\left\|l^{(i+1)}\right\|$ и вектор смещения $b^{(i)}$ размера $\left\|l^{(i+1)}\right\|$ промежуточного слоя. Совокупность наборов всех весов каждого слоя W и b обозначим как $\theta \equiv (W, b)$.

В ИНС есть два типа слоев: видимые слои и скрытые слои. Видимый слой получает входной экземпляр $\vec{x} \in X$ с дополнительной переменной смещения. Вектор числовых атрибутов $\vec{x}$ описывает экземпляр и обычно нормализуется [37]. Разница между видимым слоем и скрытыми слоями заключается в том, что видимый слой считается предварительно вычисленным и готовым для передачи второму слою.

Чтобы выполнить ИНС, $l^{(2)}$ активируется выходом $l^{(1)}$ (т. е. $\vec{x}$), взвешенным с $W^{(1)}$, затем выход $l^{(2)}$, взвешенный с $W^{(2)}$, используется для активации $l^{(3)}$ и так далее, пока не будет активирован последний слой. Этот процесс известен как прямое распространение. Пусть $\vec{h}_{w,b}^{(i)}$ будет вектором выходов нейронов слоя $l^{(i)}$. Тогда на $l^{(i+1)}$ слое

$$\vec{h}_{w,b}^{(i+1)} = f_h\left(w_h^{(i)}\vec{x}^{(i)} + \vec{b}^{(i)}\right),$$

где f — функция активации промежуточного слоя; $w_h^{(i)}$ матрица весов промежуточного слоя, а $\vec{b}^{(i)}$ вектор смещений промежуточного слоя.

Обычная функция активации представляет собой сигмовидную функцию вида $f(\vec{x}) = \frac{1}{1+e^{\vec{x}}}$. Вектор $\vec{h}_{w,b}^{(i+1)}$декодируется, чтобы воссоздать вход, так что

$$y^i = f\left(w_y^{(i)}\vec{h}_{w,b}^{(i+1)} + b_y^{(i)}\right) = \hat{x},$$

где $f_e, w_y^{(i)}, b_y^{(i)}$ сотвтствуют входному слою.

Выходная последовательность имеет такой же размер что и входная.

Под ошибкой понимается разность между входным и выходным вектором, которая должна удовлетворять соотношению $L = |x - y| = \left|x - \hat{x}\right| \rightarrow min.$

Минимизация ошибки достигается путем повторения перечисленных шагов, с помощью алгоритма обратного распространения ошибки путем обновления весов $\boldsymbol{w_y^{(i)}}$, до тех пор, пока ошибка L не снизится до приемлемого уровня. Эффект от применения АК объясняется фильтрационными свойствами, которые будут проиллюстрированы ниже.

Использование таких нейронных сетей в ряде случаев приносит хорошие результаты. Так в [38] авторы используют автокодировщики (АК) для обнаружения аномалий в электросетях. В [39] рассмотрено использование ансамбля глубоких нейронных сетей для решения задачи отслеживания объектов в режиме онлайн. В предложенном методе используется стековый АК, каждый слой которого служит в качестве отдельного пространства атрибутов для отображения исходных данных. Схема преобразует каждый слой АК в глубокую нейронную сеть, которая

используется в качестве дискриминационной модели бинарной классификации. Однако, обучение глубокой нейронной сети является сложным и не может быть практически выполнено на простом сетевом устройстве. В [40, 41] авторы предлагают использовать АК для извлечения атрибутов из наборов данных с целью улучшения обнаружения киберугроз. В [42] предложено использовать универсальный АК для обнаружения аномалий.

1.5 Потоковая классификация

В отличие от традиционных пакетных алгоритмов (offline), потоковые алгоритмы (online) классификации не могут оперировать со всем объемом данных. В результате обучение модели классификации и её тестирование осуществляются в режиме online. Известным недостатком такой классификации является то, что такой анализ «в один проход» не распознает изменений, происходящих с данными с момента начала обработки. При этом процесс классификации может потребовать одновременного построения модели и её тестирования в постоянно изменяющейся среде, в результате чего процессы тестирования и обучения «конкурируют» между собой, снижая качество классификации.

С точки зрения оценки модели классификации, потоковое обучение не сосредоточено на повторном использовании данных для получения максимальной отдачи от ограниченного объема, поскольку предполагается большое количество данных, при наличии которого точность обобщения можно измерить с помощью метода отложенной выборки. Существенная разница состоит в том, что для проверки можно выделить большой набор экземпляров для точного измерения метрик классификации.

Вместо максимального использования данных акцент смещается на тренды с течением времени: при пакетной классификации конечным результатом обучения является одна статическая модель классификации, тогда как при потоковой классификации модель развивается с течением времени и может использоваться на разных этапах роста. При пакетном обучении проблема ограниченности данных решается путем анализа и усреднения нескольких моделей, созданных с различным

случайным расположением обучающих и тестовых данных. В потоковой классификации проблема фактически неограниченного объема данных ставит другие задачи. Одно из решений предполагает создание снимков в разное время во время внедрения модели, чтобы увидеть, насколько модель улучшится при дальнейшем обучении.

Процедура оценки алгоритма классификации определяет, какие экземпляры используются для обучения алгоритма, а какие — для проверки результатов модели, выдаваемых алгоритмом. Процедура, исторически использовавшаяся при пакетном обучении, частично зависела от размера данных. Увеличение размеров данных и практические ограничения по времени делают неприменимыми процедуры, которые повторяют обучение слишком много раз. При использовании значительно больших источников данных общепринято, что необходимо уменьшить количество повторений или множеств, чтобы эксперименты могли завершиться в разумные сроки. При использовании самых больших источников данных при пакетном обучении, порядка сотен тысяч примеров и более, можно использовать одиночный прогон, поскольку это требует наименьших вычислительных усилий. Надежность, потерянная из-за отсутствия повторных запусков, компенсируется надежностью, полученной за счет огромного количества задействованных экземпляров.

Если рассматривать потоковую классификацию как частный случай пакетной классификации при большом количестве экземпляров, то для оценки модели классификации можно использовать метод отложенной выборки.

Чтобы отслеживать производительность модели с течением времени, модель можно оценивать периодически, например, после каждого миллиона обучающих экземпляров. Слишком частое тестирование модели может значительно замедлить процесс оценки, в зависимости от размера тестового набора.

Возможным источником экземпляров для отложенной выборки являются новые экземпляры из потока, которые еще не использовались для обучения алгоритма. Процедура может «заглядывать вперед», собирая из потока пакет экземпляров для использования в качестве тестовых примеров, и, если желательно

эффективное использование примеров, их можно затем передать алгоритму для дополнительного обучения после завершения тестирования. Этот метод был бы предпочтительнее в сценариях со сменой концепции, поскольку он позволяет измерить способность модели адаптироваться к последним трендам в данных.

Когда не предполагается смена концепта, достаточно одного статического отложенного набора, что позволяет избежать проблемы различий в оценках между потенциальными тестовыми выборками. Предполагая, что тестовый набор независим и достаточно велик по сравнению со сложностью целевой концепции, он обеспечит точное измерение метрики обобщения.

Альтернативная схема оценки алгоритмов потоковой классификации — чередование тестирования с обучением. Каждый отдельный пример можно использовать для тестирования модели перед его использованием для обучения, что позволяет постепенно повышать точность. При этом модель всегда тестируется на примерах, которых она еще не видела. Преимущество этой схемы состоит в том, что для тестирования не требуется никакого дополнительного набора, что позволяет максимально использовать доступные данные. Это также обеспечивает плавный график метрик с течением времени, поскольку каждый отдельный пример будет становиться все менее значимым по сравнению с общим средним значением.

Недостатком этого подхода является то, что он затрудняет точное разделение и измерение времени обучения и тестирования. Кроме того, истинная точность, которую алгоритм может достичь в данный момент, скрыта — алгоритмы будут наказываться за ранние ошибки независимо от уровня точности, на который они в итоге способны, хотя этот эффект со временем будет уменьшаться.

С помощью этой процедуры статистика обновляется с каждым примером в потоке и при желании может быть записана с таким уровнем детализации. Из соображений эффективности можно использовать параметр выборки, чтобы уменьшить требования к хранению результатов путем записи только через определенные промежутки времени, как при методе отложенной выборки.

Широкое распространение получили алгоритмы потоковой классификации Adaptive Random Forests [43], Hoeffding Adaptive Tree, K nearest neighbors, Oza Bagging [44].

1.6 Смена концепта

Исследования, проводимые в области машинного обучения, показывают, что неожиданные изменения в распределении данных делают обученную модель ненадежной. Эта ненадежность является основным препятствием для широкой адаптации технологии машинного обучения в реальной жизни. Чтобы сделать модель машинного обучения обновляемой и надежной, необходимо включить механизм обнаружения смены концепта.

Методы обнаружения смены концепта могут быть классифицированы по типу механизма обнаружения на контролируемые [45], неконтролируемые [46] и полуконтролируемые. Существуют также активные и пассивные методы обнаружения смены концепта [47].

1.6.1 Контролируемые методы обнаружения смены концепта

В сценариях с контролируемым обнаружением смены концепта и контролируемым машинным обучением предполагается, что метки классов доступны сразу после предсказания, а смена концепта обнаруживается на основе производительности классификатора. Если достоверность или другие показатели оценки, такие как точность и полнота, падают ниже порога в рамках определенного размера окна, то считается, что произошла смена концепта. Это предположение относительно доступности истинных меток не реалистично, и в большинстве случаев истинные метки не доступны сразу после предсказания [46]. В таких ситуациях модели обнаружения смены концепта теряют свою значимость. Контролируемые методы обнаружения смены концепта могут обнаружить только реальный дрейф. Этот метод также известен как обнаружение смены концепта на основе коэффициента ошибок. Контролируемые методы обнаружения смены

концепта можно разделить на статистические, оконные и ансамблевые (на основе блоков и на основе инкрементов) [45].

Статистические методы обнаружения дрейфа применяют статистические тесты к окну оценок работы классификатора, чтобы обнаружить любые значительные изменения в его работе. К статистическим методам относят последовательный тест соотношения вероятностей (SPRT) [48], кумулятивная сумма (CUSUM) [49], тест Пейджа-Хинкли (PH) [50], Stagger [51], метод обнаружения дрейфа (DDM) [52], метод раннего обнаружения (EDDM) [53], метод обнаружения дрейфа для онлайн-обучения дисбалансу классов (DDM-OCI) [54], Statistical Tests for Equal Proportions (STEPD) [55], метод обнаружения дрейфа на основе адаптивного окна (ADWIN2) [56], метод обнаружения дрейфа с использованием границ Хеффдинга (HDDM) [57], четверка линейных коэффициентов (LFR) [58], Hierarchical Linear Four Rates (HLFR) [59], Reactive Drift Detection Method (RDDM) [60], обнаружение смены концепта на основе точного теста Фишера – FPDD (Fisher Proportion Drift Detector), Fisher-based Statistical Drift Detector (FSDD) и Fisher Test Drift Detector (FTDD) [61], The McDiarmid Drift Detection Methods (MDDMs) [62].

Вместо того чтобы использовать частоту ошибок одного классификатора для обнаружения смены концепта, ансамблевые методы используют группу классификаторов и их среднюю частоту ошибок для обнаружения таких изменений. Поскольку использование ансамблевых методов в машинном обучении оказалось более эффективным по сравнению с одиночным классификатором, исследовательское сообщество приложило значительные усилия для обнаружения дрейфа в работе ансамбля и для принятия решения о том, как и когда обновлять ансамбль, чтобы учесть изменения, произошедшие в распределении данных, что привело к изменению работы ансамбля. К ансамблевым методам относятся алгоритм потокового ансамбля (SEA) [63], Accuracy Weighted Ensemble (AWE) [64], Accuracy Updated Ensemble (AUE) [65], метод инкрементного ансамбля на основе динамически взвешенного большинства (DWM) [66], Learn++ [67], Learn++.MT [68], Learn++.NC (New Class) [69], Learn++NSE [70], Learn++NIE (Non-

Stationary and Imbalanced Environments) [71], Learn++.CDS [72]. Было обнаружено, что производительность алгоритмов серии Learn++ сильно зависит от базовых моделей [73].

1.6.2 Полуконтролируемые методы обнаружения смены концепта

Зависимость методов обнаружения смены концепта от наличия меток классов в контролируемых методах обнаружения смены концепта и связанные с этим затраты и задержки в получении истинных меток в реальных приложениях привлекли внимание исследовательского сообщества к методам построения структуры, которые либо не зависят от меток классов, либо зависят в ограниченной степени. Некоторые из этих работ требуют маркированных данных для начального обучения классификаторов и инициализации методов обнаружения смены концепта, а уровни доверия классификатора нужны только для обнаружения смены концепта. Эти методы были классифицированы как полуконтролируемые методы обнаружения смены концепта [74]. Полуконтролируемые методы обнаружения смены концепта требуют ограниченного количества маркированных данных для первоначального обучения классификатора или ансамбля. Смена концепта обнаруживается на основе изменений в уровне доверия к предсказанию. Для обновления модели требуются метки только для тех случаев, когда уровень доверия низкий. К полуконтролируемым методам обнаружения смены концепта относятся Semi-Supervised Adaptive Novel Class Detection (SAND) [75], ECHO (Efficient Handling of Concept Drift Evolution over Stream Data) [76], метод обнаружения смены концепта, использующий самоаннотацию и ансамбль базовых моделей (подобно SAND) [77], Online Novelty and Drift Detection Algorithm (OLINDA) [78].

1.6.3 Неконтролируемые методы обнаружения смены концепта

Ограничения в контролируемых и полуконтролируемых методах обнаружения смены концепта, такие как зависимость от меток классов или уровней

доверия классификатора, подтолкнули исследовательское сообщество к использованию неконтролируемых методов обнаружения смены концепта, направленных на выявление смены концепта путем мониторинга изменений в распределении данных [46]. Алгоритмы неконтролируемых методов обнаружения дрейфа обычно поддерживают два окна, а именно: эталонное (историческое) окно и окно обнаружения (новые данные), и используют меру расстояния для количественной оценки разницы между распределением исторических данных и новых данных. Историческое окно остается фиксированным, в то время как окно обнаружения является скользящим. Если разница в распределении данных в двух окнах значительна, то обнаруживается дрейф с указанием точек дрейфа [79]. Неконтролируемое обнаружение дрейфа также известно как "обнаружение дрейфа на основе распределения данных". Неконтролируемые методы обнаружения смены концепта рассматриваются в [80], [81], [79] и [74]. Большинство неконтролируемых методов обнаружения смены концепта поддерживают эталонное окно, содержащее экземпляры данных, на которых был обучен последний классификатор, и окно обнаружения, которое рассматривается с точки зрения обнаружения смены концепта. Если смена концепта обнаруживается на основе пакета элементов данных в окне обнаружения, то эти методы называются методами обнаружения смены концепта на основе пакета, а если смена концепта обнаруживается на основе каждого отдельного экземпляра в окне обнаружения, то они называются методами обнаружения смены концепта в режиме онлайн.

Неконтролируемые методы обнаружения смены концепта, основанные на пакетном подходе, выявляют смену концепта на основе изменений в распределении данных в пакете фиксированного или динамического размера. Некоторые из этих методов используют весь пакет для обнаружения, в то время как другие используют часть пакета или подмножество образцов в окне обнаружения. К таким методам относятся Margin Density Drift Detection (MD3) [82] и его модификацию [83], потоковая структура predict-detect – механизм обнаружения смены концепта, вызванную действиями злоумышленников, для обнаружения атак противника на классификатор или систему обнаружения смены концепта [84],

метод обнаружения дрейфа без наблюдения на основе активного обучения, не требующий маркированных образцов для обнаружения смены концепта [85], [86], неконтролируемый метод обнаружения смены концепта в распознавании активности U_{Detect} [87], метод обнаружения смены концепта на основе несходства показателей плотности [88], метод быстрого и точного обнаружения аномалий (FAAD) [89], SQSI (Stream Quantification by Score Inspection) [90] и его расширение [91].

В отличие от пакетных методов, которые накапливают экземпляры в пакет определенного размера и затем используют некоторый алгоритм обнаружения смены концепта для сравнения текущего пакета с некоторым эталонным пакетом, онлайн методы выполняют обнаружение смены концепта при поступлении каждого отдельного экземпляра. Для инициализации эти методы должны накопить некоторое количество экземпляров в самом начале. После инициализации поддерживается эталонное окно, которое может быть фиксированным на обучающих экземплярах или скользящим по входящему потоку данных. Другое скользящее окно определяется по входящему потоку данных и называется окном обнаружения. Для обнаружения смены концепта распределение эталонного окна и окна обнаружения сравниваются, и выполняется статистический тест для проверки значимость сходства или несходства. На основе опорного окна - фиксированного или скользящего, онлайн методы могут быть классифицированы как с фиксированным опорным окном или скользящим опорным окном.

Онлайн-методы обнаружения смены концепта, основанные на фиксированном опорном окне, поддерживают опорное окно (основанное на данных обучения) неизменным. К таким методам относятся метод обнаружения дрейфа на основе инкрементального теста Колмогорова-Смирнова (IKS-bdd) [92], CD-TDS (Change Detection in Transactional Data Streams) [93].

В подходах с использованием скользящего опорного окна для онлайн обнаружения смены концепта опорное окно перемещается по входящему потоку данных. К таким методам относятся модифицированная версия теста PageHinkley OMV-PHT (Online Modified Version of Page Hinkley Test) [94], непараметрический

многомерный метод обнаружения дрейфа (NM-DDM) [95], Plover [96], обнаружение смены концепта на основе распределения (DbDDA) [97].

1.6.4 Методы обнаружения смены концепта на основе глубокого обучения

Некоторые недавние работы по обнаружению смены концепта основаны на методах глубокого обучения, включая автокодировщики и ограниченную машину Больцмана (RBM). В [98] использовали байесовские автокодировщики для обнаружения смены концепта в данных датчиков в промышленной среде. Для обнаружения были использованы три различные меры, включая потери при реконструкции, алеаторные и эпистемические неопределенности. В случае реального дрейфа (смена концепта уже присутствует в данных из-за ухудшения состояния датчиков) все три меры показывают значительное отклонение. В данной работе, однако, обнаружение смены концепта происходит без учета наличия и влияния различных классов в данных.

В [99] применили RBM на синтетическом бинарном наборе данных, сгенерированном с помощью RBM, для обнаружения внезапной и постепенной смены концепта. Для обнаружения были использованы два показателя - потери при реконструкции и свободная энергия. В случае смены концепта оба показателя указывают на значительное отличие от нормальных данных. В [100] применили автокодировщики к тому же набору данных и использовали ошибку реконструкции и перекрестную энтропию с помощью автокодировщиков и доказали, что внезапная и постепенная смена концепта может быть обнаружена с помощью автокодировщиков. Однако в обеих работах обнаружение происходит без контроля, без учета влияния изменений в распределении данных на границы классификатора.

Все три вышеперечисленные методики сосредоточены на виртуальном дрейфе, т.е. на обнаружении изменений в распределении данных без учета влияния изменений на классификатор. Обнаружение смены концепта на основе автокодировщика (ADD) [101] - еще одна недавняя работа, в которой для

обнаружения смены концепта в фишинговых данных используются автокодировщики. Для обнаружения потери при реконструкции сравниваются с заданными пользователем пороговыми значениями, а в качестве классификатора используется дерево Хеффдинга. Автор показал значительное улучшение точности после обнаружения смены концепта и адаптации. Однако эта работа также имеет некоторые ограничения, такие как использование одного автокодировщика для моделирования распределения всех классов в наборе данных классификации, использование одного порога для всех наборов данных и отсутствие учета возможности ложных срабатываний.

Выводы

Подходы на основе глубокого обучения могут работать независимо от наличия истинных меток для обнаружения смены концепта. Эти методы также не зависят от классификатора, используемого для прогнозирования в сценариях контролируемого машинного обучения, и обладают неотъемлемой способностью обобщать многомерные данные.

Модель обнаружения смены концепта должна быть такой же надежной, как и модель, основанная на контролируемых методах обнаружения смены концепта. Процесс обнаружения смены концепта не должен зависеть от меток классов, как в случае с неконтролируемыми методами обнаружения дрейфа. При обучении модели обнаружении смены концепта следует использовать все имеющиеся метки, как в полуконтролируемых методах обнаружения смены концепта. Такой механизм, как прогнозирование достоверности, должен быть включен без ущерба для различных уровней достоверности с разными классификаторами. Возможности глубокого обучения следует использовать для обобщения данных высокой размерности.

ГЛАВА 2 КЛАССИФИКАЦИЯ ПРОТИВОПРАВНЫХ, ВРЕДОНОСНЫХ И НЕЖЕЛАТЕЛЬНЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ В РЕЖИМЕ OFFLINE В УСЛОВИЯХ НАЛИЧИЯ ФОНОВОГО ТРАФИКА

2.1 Постановка задачи

Необходимость обнаружения использования определённых мобильных приложением объясняется требованиям законодательства Российской Федерации [1] в определённых случаях блокировать использование приложений. Причиной блокировки может стать распространение информации с нарушением авторских и (или) смежных прав, а также не выполнением требования, предъявляемых к организаторам распространения информации в сети Интернет.

Второй причиной необходимости обнаружения приложений является быстро растущее число мобильных приложений, являющихся вредоносными или использующиеся злоумышленниками в своей деятельности.

Большинство мобильных приложений используют доступ в сеть, то есть генерируют сетевой трафик. При помощи анализа сетевого трафика можно выделить в потоке трафик определённых приложений (являющиеся вредоносными, нарушающими авторские и (или) смежные права либо не выполняющие требования федерального законодательства [1]).

Задача выделения такого трафика с целью установления факта использования определённого приложения является задачей многоклассовой классификации, которая может быть решена с применением методов машинного обучения.

Задача классификации в потоке сетевого трафика определённых приложений имеет особенность – наличие большого количества фонового трафика, генерируемого приложениями, которые не представляют интерес. Кроме того, сетевой трафик может меняться со временим, что может привести к снижению метрик классификации. Необходимы алгоритмы, способные обнаружить такое изменение и провести повторное обучение моделей.

Задача классификации мобильных приложений на основе анализа сетевого трафика формулируется следующим образом. Пусть $P = (p_i)_{i=1}^{M_{IP}} = (p_1, p_2, \dots, p_{M_{IP}})$

последовательно поступающие IP-датаграмы, где $p_i = \left(b_j\right)_{j=1}^{N_{IP}}$ – последовательность байтов размера N_{IP}.

Для нахождения в последовательности IP-датаграм TCP-сеансов и вычисления атрибутов сеанса требуется получить значения полей заголовков IP-датаграм и полезную нагрузку.

Для этого сначала введём в рассмотрение функцию получения заголовка и полезной нагрузки IP датаграммы:

$$ExtractIpHeaderAndPayload: \{p_i, p_i \in P\} \rightarrow \left\{\left(p_i^{IP_{header}}, p_i^{IP_{payload}}\right)\right\} \quad (2.1)$$

где $p_i = \left(b_j\right)_{j=1}^{N_{IP}}$ – IP-датаграма, представленная в виде последовательности байтов;

$p_i^{IP_{header}} = \left(b_j\right)_{j=1}^{N_{IP}^{header}} = \left(b_j^{IP_{header}}\right)_{j=1}^{N_{IP}^{header}}$ – последовательность байтов длины N_{IP}^{header}, являющаяся заголовком IP-датаграмы;

$p_i^{IP_{payload}} = \left(b_j\right)_{j=N_{IP}^{header}+1}^{N_{IP}} = \left(b_j^{IP_{payload}}\right)_{j=1}^{N_{IP}^{payload}}$ – последовательность байтов длины $N_{IP}^{payload} = N_{IP} - N_{IP}^{header}$, являющаяся полезной нагрузкой IP-датаграмы.

Для получения значения полей заголовка IP-датаграмы ввёдем следующие функции.

Функция получения IP-адреса источника:

$$IpSrc: \left\{p_i^{IP_{header}}\right\} \rightarrow \{k\}_{k=0}^{2^{32}-1} \quad (2.2)$$

где $p_i^{IP_{header}}$ – последовательность байтов заголовка IP-датаграмы;

k – IP-адрес источника, представленный в виде 32-битного целого не отрицательного числа.

Функция получения IP-адреса назначения:

$$IpDst: \left\{p_i^{IP_{header}}\right\} \rightarrow \{k\}_{k=0}^{2^{32}-1} \quad (2.3)$$

где $p_i^{IP_{header}}$ – последовательность байтов заголовка IP-датаграмы;

k – IP-адрес назначения, представленный в виде 32-битного целого неотрицательного числа.

Функция определения протокола, вложенного в IP-датаграму, определяется следующим образом:

$$IpProtocol: \{p_i^{IP_{header}}\} \to \{k\}_{k=0}^{2^8-1} \quad (2.4)$$

где $p_i^{IP_{header}}$ – последовательность байтов заголовка IP-датаграмы;

k – номер протокола, представленный в виде 8-битного целого неотрицательного числа.

Для протокола TCP значение функции будет равно 6.

Функция определения полной длины IP-датаграмы имеет вид:

$$IpTotalLen: \{p_i^{IP_{header}}\} \to \{k\}_{k=20}^{2^{16}-1} \quad (2.5)$$

где $p_i^{IP_{header}}$ – последовательность байтов заголовка IP-датаграмы;

k – длина IP-датаграмы с учётом заголовка, представленная в виде 16-битного целого числа, $k > 20$.

Функция определения длины полезной нагрузки IP-датаграмы:

$$IpPayloadLen: \{p_i^{IP_{header}}\} \to \{k\}_{k=0}^{2^{16}-20} \quad (2.6)$$

где $p_i^{IP_{header}}$ – последовательность байтов заголовка IP-датаграмы;

k – длина полезной нагрузки IP-датаграмы, представленная в виде 16-битного целого неотрицательного числа, $k < 2^{16} - 20$.

Для нахождения в последовательности IP-датаграм TCP-сеансов и вычисления атрибутов сеанса также требуется получить значение полей заголовков TCP-сегментов.

Функция получения заголовка и полезной нагрузки TCP-сегмента:

$$ExtractTcpHeaderAndPayload: \left\{p_i^{IP_{payload}}\right\} \to \left\{\left(p_i^{TCP_{header}}, p_i^{TCP_{payload}}\right)\right\} \quad (2.7)$$

где $p_i^{IP_{payload}}$ – полезная нагрузка IP-датаграмы;

$p_i^{TCP_{header}} = \left(b_j^{IP_{payload}}\right)_{j=1}^{N_{TCP}^{header}} = \left(b_j^{TCP_{header}}\right)_{j=1}^{N_{TCP}^{header}}$ – последовательность байтов длины N_{TCP}^{header}, являющаяся заголовком TCP-сегмента;

$$p_i^{TCP_{payload}} = \left(b_j^{IP_{payload}}\right)_{j=N_{TCP}^{header}+1}^{N_{IP}^{payload}} = \left(b_j^{TCP_{payload}}\right)_{j=1}^{N_{TCP}^{payload}}$$

последовательность байтов длины $N_{TCP}^{payload} = N_{IP}^{payload} - N_{TCP}^{header}$, являющаяся полезной нагрузкой TCP-сегмента.

Для получения значений полей заголовка TCP-сегмента ввёдем следующие функции.

Функция получения порта источника:

$$TcpPortSrc: \left\{p_i^{TCP_{header}}\right\} \rightarrow \{k\}_{k=0}^{2^{16}-1} \quad (2.8)$$

где $p_i^{TCP_{header}}$ – последовательность байтов заголовка TCP-сегмента;

k – порт источника, представленный в виде 16-битного целого неотрицательного числа.

Функция получения порта назначения:

$$TcpPortDst: \left\{p_i^{TCP_{header}}\right\} \rightarrow \{k\}_{k=0}^{2^{16}-1} \quad (2.9)$$

где $p_i^{TCP_{header}}$ – последовательность байтов заголовка TCP-сегмента;

k – порт назначения, представленный в виде 16-битного целого неотрицательного числа.

Функция определения наличия флага SYN:

$$TcpFlagSyn: \left\{p_i^{TCP_{header}}\right\} \rightarrow \{0,1\} \quad (2.10)$$

где $p_i^{TCP_{header}}$ – последовательность байтов заголовка TCP-сегмента.

Значением функции является 1, если флаг SYN установлен, 0 в противном случае.

Функция определения наличия флага SYN:

$$TcpFlagFin: \left\{p_i^{TCP_{header}}\right\} \rightarrow \{0,1\} \quad (2.10)$$

где $p_i^{TCP_{header}}$ – последовательность байтов заголовка TCP-сегмента.

Значением функции является 1, если флаг FIN установлен, 0 в противном случае.

Функция получения дины полезной нагрузки TCP-сегмента:

$$TcpPayloadLen: \left\{p_i^{TCP_{header}}\right\} \rightarrow \{k\}_{k=0}^{2^{32}-1} \quad (2.11)$$

где $p_i^{TCP_{header}}$ – последовательность байтов заголовка TCP-сегмента.

k – длина полезной нагрузки TCP-сегмента, представленная в виде 32-битного целого неотрицательного числа.

Отфильтруем поток IP-датаграм P таким образом, чтобы в нём содержались только датаграмы, содержащие в качестве полезной нагрузки TCP-сегменты и выделим из байтов IP-датаграмы следующие поля:

- IP-адрес источника;
- IP-адрес назначения;
- порт источника;
- порт назначения;
- признак начала TCP соединения;
- признак завершения TCP соединения;
- длина IP-датаграмы;
- длина полезной нагрузки IP-датаграмы;
- длина полезной нагрузки TCP-сегмента.

Функция фильтрации и получения заголовков IP-датаграмы и TCP-сегмента

$$FilterTcp(\mathrm{P}) = \\ = \left((b_{header}^{IP}, b_{header}^{TCP}, b_{payload}^{TCP}) | IpProtocol(b_{header}^{IP}) = 6\right), \quad (2.12)$$

где P – последовательность IP-датаграм, $p_i \in P$;

$b_{header}^{IP} = [ExtractIpHeaderAndPayload(p_i)]_1$;

$b_{payload}^{IP} = [ExtractIpHeaderAndPayload(p_i)]_2$;

$b_{header}^{TCP} = \left[ExtractTcpHeaderAndPayload\left(b_{payload}^{IP}\right)\right]_1$;

$b_{payload}^{TCP} = \left[ExtractTcpHeaderAndPayload\left(b_{payload}^{IP}\right)\right]_2$.

Функция получения значений полей заголовков IP-датаграммы и TCP-сегментов:

$$ExtractFields\left(b_{header}^{IP}, b_{header}^{TCP}, b_{payload}^{TCP}\right) = \\ = (IpSrc(b_{header}^{IP}), IpDst(b_{header}^{IP}), TcpPortSrc\left(b_{header}^{TCP}\right), TcpPortDst\left(b_{header}^{TCP}\right), \\ TcpFlagSyn\left(b_{header}^{TCP}\right), TcpFlagFin\left(b_{header}^{TCP}\right), IpTotalLen(b_{header}^{IP}), \\ IpPayloadLen(b_{header}^{IP}), TcpPayloadLen\left(b_{header}^{TCP}\right)) \quad (2.13)$$

Отфильтрованные IP-датаграмы:

$$P^{TCP} = (\mathrm{E}xtractFields([p_i]_1, [p_i]_2, [p_i]_3) | p_i \in FilterTcp(P)) \quad (2.14)$$

Сгруппируем IP-датаграммы, принадлежащие одному TCP-сеансу с помощью функции:

$$TcpSessions: \{P^{TCP}\} \rightarrow \{S\} \quad (2.15)$$

где P^{TCP}- последовательность значений полей IP-датаграм, полученная в (2.14)

$S = (s_i)_{i=1}^{M}$ – поток TCP-сеансов

Тогда последовательность TCP-сеансов:

$$S = TcpSessions(P^{TCP}) \quad (2.16)$$

Пусть все сеансы в последовательности S принадлежат множеству мобильных приложений $A = \{a_k\}_{k=1}^{K}$.

Необходимо представить последовательность TCP-сеансов в виде набора атрибутов X, набора правильных ответов Y и построить модели классификации C и модель обнаружения смены концепта D:

$$C: \{X\} \to \{Y\} \quad (2.17)$$

$$D: \{X\} \times \{Y\} \to \{0,1\} \quad (2.18)$$

Отличительной особенностью решаемых задач являются условия априорной неопределенности о составе, характеристиках, количестве классифицируемых приложений, а также наличия смены концепта, когда статистические свойства класса или целевой переменной изменяются со временем случайным образом.

2.2 Сбор данных

Для формирования наборов данных на мобильных устройствах под управлением ОС Android с помощью Android VPN API [102-105] осуществлялся сбор необработанных данных сетевого трафика в виде IP-пакетов. Обработка данных (в том числе фильтрация пакетов, содержащих данные протокола TCP, группировка пакетов в TCP сеансы, вычисление атрибутов TCP сеансов, характеризующих особенности анализируемых приложений) осуществлялась на сервере всякий раз, когда поступал IP-пакет. С использованием разработанного программного комплекса ПК CAT был собран сетевой трафик трёх типов мобильных приложений: «с шифрованием», «без шифрования», «с частичным шифрованием». Всего собрано 92 334 потока и 6 989 991 пакет сетевого трафика для 18 мобильных приложений.

Приложения можно разделить на три группы:

1. Приложения, которые шифруют сетевой трафик для всех соединения;

2. Приложения, которые не шифруют сетевой трафик;
3. Приложения, которые шифруют сетевой трафик определённых соединений.

Информация о собранных приложениях и потоках сетевого трафика представлена в таблицах 2.1–2.2.

Таблица 2.1 – Информация о собранных приложениях

№	ID	Сокращенное название	Имя пакета	Название приложения	Шифрование
a_1	13	F_RE	com.kirik88.fireader	Фишки с FiReader – юмор, демотиваторы, посты	нет
a_2	61	GVL	ru.godville.android	Годвилль	нет
a_3	58	MK	com.mobilein.mk	Московский комсомолец	нет
a_4	82	NTV	ru.ntv.client	НТВ: новости, видео, передачи	нет
a_5	48	PZ_EPR	ru.itsilver.pizzaempire	ПиццаСушиВок - доставка еды	нет
a_6	23	WR	com.wolfram.android.alpha	WolframAlpha	нет
a_7	14	HSN	com.blizzard.wtcg.hearthstone	Hearthstone	да
a_8	3	ISG	com.instagram.android	Instagram *(проект Meta Platforms Inc., деятельность которой запрещена в Российской Федерации)*	да
a_9	9	MI_RU	ru.mail.mailapp	Почта Mail.Ru	да
a_{10}	17	PKB	ru.pikabu.android	Пикабу – юмор и новости	да
a_{11}	12	SB	ru.sberbankmobile	Сбербанк Онлайн	да

№	ID	Сокращенное название	Имя пакета	Название приложения	Шифрование
a_{12}	10	SP	com.skype.raider	Скайп – бесплатные мгновенные сообщения и видеозв.	да
a_{13}	2	4PDA	ru.fourpda.client	4PDA	частично
a_{14}	18	BD	com.badoo.mobile	Badoo – Новые знакомства	частично
a_{15}	46	BK	com.booking	Booking.com бронь отелей	частично
a_{16}	5	GG_CM	com.android.chrome	Google Chrome: быстрый браузер	частично
a_{17}	11	KMS	com.nsadv.kommersant	Коммерсантъ	частично
a_{18}	57	YD_BS	com.yandex.browser	Яндекс.Браузер – с Алисой	частично

Таблица 2.2 – Информация о собранных потоках сетевого трафика

№	Сокращенное название	Количество потоков	Количество пакетов
a_1	F_RE	5422	576581
a_2	GVL	5016	61343
a_3	MK	5335	107202
a_4	NTV	5908	233982
a_5	PZ_EPR	5097	64460
a_6	WR	5190	61140
a_7	HSN	5028	227688
a_8	ISG	4979	1916363
a_9	MI_RU	5078	246184
a_{10}	PKB	5329	265071
a_{11}	SB	5110	241235
a_{12}	SP	5244	232510
a_{13}	4PDA	4974	524215
a_{14}	BD	4976	581212
a_{15}	BK	5326	552606
a_{16}	GG_CM	3865	620277
a_{17}	KMS	5325	338327
a_{18}	YD_BS	5132	139595

2.3 Вычисление атрибутов

Представим каждый TCP сеанс в виде последовательности, состоящей из 21 атрибута, вычисляемые с помощью функций f_i, представленные в таблице 2.3.

Таблица 2.3 – Функции вычисления атрибутов TCP сеанса

№	ID	Сокращенное название	Название
f_1	1	L3_Tot_Pl_Sz_C2S	Общий объем полезной нагрузки на сетевом уровне от клиента в байтах. Не включает длину заголовка IP
f_2	2	L3_Tot_Pl_Sz_S2C	Общий объем полезной нагрузки на сетевом уровне от сервера в байтах. Не включает длину заголовка IP
f_3	3	L4_Tot_Pl_Sz_C2S	Общий объем полезной нагрузки на транспортном уровне от клиента в байтах. Не включает длину заголовка IP и TCP (UDP)
f_4	4	L4_Tot_Pl_Sz_S2C	Общий объем полезной нагрузки на транспортном уровне от сервера в байтах. Не включает длину заголовка IP и TCP (UDP)
f_5	5	L3_Avg_Dtg_Sz_C2S	Средний размер пакета на сетевом уровне со стороны клиента в байтах
f_6	6	L3_Avg_Dtg_Sz_S2C	Средний размер пакета на сетевом уровне со стороны сервера.
f_7	7	L4_Avg_Pl_Sz_C2S	Средний размер полезной нагрузки на транспортном уровне со стороны клиента.
f_8	8	L4_Avg_Pl_Sz_C2S	Средний размер полезной нагрузки на транспортном уровне со стороны сервера.
f_9	9	L3_Std_Tot_Sz_C2S	Стандартное отклонение размера пакета на сетевом уровне со стороны клиента
f_{10}	10	L3_Std_Tot_Sz_S2C	Стандартное отклонение размера пакета на сетевом уровне со стороны сервера
f_{11}	11	L4_Std_Pl_Sz_C2S	Стандартное отклонение размера полезной нагрузки на транспортном уровне со стороны клиента
f_{12}	12	L4_Std_Pl_Sz_S2C	Стандартное отклонение размера полезной нагрузки на транспортном уровне со стороны сервера
f_{13}	13	L3_Avg_Pac4Msg_C2S	Среднее число пакетов сетевого уровня для передачи сообщения со стороны клиента
f_{14}	14	L3_Avg_Pac4Msg_S2C	Среднее число пакетов сетевого уровня для передачи сообщения со стороны сервера
f_{15}	15	L3_Efficiency_C2S	КПД клиента - Количество переданной полезной нагрузки транспортного уровня, делённое на общее количество переданной нагрузки транспортного и сетевого уровня со стороны клиента.
f_{16}	16	L3_Efficiency_S2C	КПД сервера - Количество переданной нагрузки сетевого уровня, делённое на общее количество переданной нагрузки транспортного и сетевого уровня со стороны сервера.
f_{17}	17	L3_Tot_Dtg_Sz_CS_ratio	Соотношение байт - во сколько раз клиент передал больше пакетов в байтовом представлении, чем сервер.
f_{18}	18	L4_Tot_Pl_Sz_CS_ratio	Соотношение полезной нагрузки транспортного уровня - во сколько раз клиент передал больше байт

№	ID	Сокращенное название	Название
			полезной нагрузки на транспортном уровне, чем сервер.
f_{19}	19	L3_Tot_Dtg_Cnt_CS_ratio	Соотношения общего количества пакетов на сетевом уровне - во сколько раз клиент передал больше пакетов на сетевом уровне, чем сервер.
f_{20}	20	L3_Tot_Dtg_Cnt_C2S	Общее количество переданных датаграмм на сетевом уровне со стороны клиента
f_{21}	21	L3_Tot_Dtg_Cnt_S2C	Общее количество переданных датаграмм на сетевом уровне со стороны сервера
f_{22}	22	L3_Src_Addr	IP-адрес источника
f_{23}	23	L3_Dst_Addr	IP-адрес назначения

Тогда набор признаков может быть вычислен с помощью функции:

$$ExtractFeatures(S) = ((f_n(s_i))_{n=1}^{N})_{i=1}^{M}, \quad (2.19)$$

где S – поток TCP-сеансов, $s_i \in S$

$N = 21$ – количество атрибутов,

M – количество элементов в последовательности TCP сеансов.

$$X = ExtractFeatures(S) \quad (2.20)$$

Каждому элементу в последовательности X сопоставим метку приложения, которому принадлежит TCP сеанс:

$$Y = (y_i = k | a_k \in A)_{i=1}^{M} \quad (2.21)$$

В качестве атрибутов классификации были выбраны статистические характеристики потока сетевого трафика и IP-адреса источника и назначения. Разделение данных

Для обучения и тестирования модели классификации необходимо разбить набор данных на 3 стратифицированные выборки: обучающую, валидационную и тестовую.

Введём функцию разделения выборки на группы:

$$Stratify(X, Y) = (X^{y_k})_{k=1}^{K} \quad (2.22)$$

где X – исходная выборка;

Y – признак, по которому производится разделение;

$X^{y_k} = (x_i | y_i = y_k, x_i \in X, y_i \in Y, y_k \in Y))$ – часть выборки X, для которой значение равно y_k;

K – количество уникальных элементов в Y.

Введём функцию случайной перестановки элементов:

$$RandomShuffle: \{Z\} \rightarrow \{\hat{Z}\} \quad (2.23)$$

где Z – последовательность элементов.

$\hat{Z}$ – новая последовательность, полученная из исходной путём случайной перестановки.

Введём функцию перестановки элементов:

$$Shuffle(Z, Indexes) = (Z_i)_{i \in Indexes} \quad (2.23)$$

где Z – последовательность элементов.

$Indexes$ –последовательность индексов (порядок перестановки), $|Z| = |Indexes|$.

Введём в рассмотрение функцию разделения последовательности:

$$SplitIndex(M, p) = \lfloor Mp \rfloor \quad (2.24)$$

где M – размер выборки;

$p \in [0; 1]$ – доля выборки;

$\lfloor Mp \rfloor$ – индекс в последовательности, удовлетворяющий критерию разбиения.

Тогда функция разбиения будет иметь вид:

$$StratifySplit(X, Y, p) = (X^{(1)}, Y^{(1)}, X^{(2)}, Y^{(2)}), \quad (2.25)$$

где X – разделяемый набор данных;

Y – набор данных с метками;

p – доля элементов в первой выборке;

$X^{(1)} = X_1^{(Y)(1)}$ – первая выборка;

$Y^{(1)} = X_2^{(Y)(1)}$ – метки для первой выборки;

$X^{(2)} = X_1^{(Y)(2)}$ – вторая выборка;

$Y^{(2)} = X_2^{(Y)(2)}$ – метки для второй выборки;

$$X^{(Y)(1)} = RandomShuffle(\left(\left(x_i^{(y_k)}, y_k\right) \middle| i \leq Split(\left|X^{(y_k)}\right|, p)\right)_{X^{(y_k)} \in X^{(Y)}})$$

$$X^{(Y)(2)} = RandomShuffle(\left(\left(x_i^{(y_k)}, y_k\right) \middle| i > Split(\left|X^{(y_k)}\right|, p)\right)_{X^{(y_k)} \in X^{(Y)}})$$

$$X^{(Y)} = Stratify\big(Shuffle(X, indexes), Shuffle(X, indexes)\big)$$

$$indexes = RandomShuffle((i)_{i=1}^{M})$$

Разделим набор данных. Выделим сначала обучающую выборку:

$$X_{train}, Y_{train}, X_{valtest}, Y_{valtest} = StratifySplit(X, Y, 0.9) \quad (2.25)$$

Затем оставшуюся часть разделим поровну:

$$X_{valid}, Y_{valid}, X_{test}, Y_{test} = StratifySplit(X_{valtest}, Y_{valtest}, 0.5) \quad (2.26)$$

В результате получили 3 выборки: обучающая выборка X_{train}, Y_{train}, составляющая 90% исходной выборки, валидационная выборка X_{valid}, Y_{valid}, составляющая 10% исходной выборки, и тестовая выборка X_{test}, Y_{test}, составляющая 10% исходной выборки.

2.4 Выбор алгоритмов и метрик оценки качества классификации

Для классификации приложений используются известные алгоритмы машинного обучения: Logistic Regression (LR), KNN, Decision Tree (DT), Random Forest (RF) и Gradient Boosting (GB), Naive Bayes (NB), C4.5, AdaBoost, SVM, ИНН.

Для оценки качества классификации с целью выбора лучшей модели классификации используется валидационный набор данных. Для финальной оценки качества выбранной модели используется тестовый набор данных.

Качество классификации оценивается с помощью следующих метрик:

- Accuracy (достоверность) – доля верно классифицированных потоков из всех;
- $Precision$ (точность) – доля верно определенных потоков класса из всех потоков, которые были отнесены к этому классу;
- Recall (полнота) – доля верно предсказанных потоков из всех, которые были определены в этот класс;
- $F1$-мера – гармоническое среднее между Recall и Precision.

Указанные метрики рассчитываются по формулам:

$$Accuracy = \frac{\sum_{i=1}^{n} TP_{ii}}{\sum_{i=1}^{n} TP_{ii} + \sum_{i=1}^{n} \sum_{j=1}^{n} F_{ij}}; \quad (2.27)$$

$$Precision_i = \frac{TP_{ii}}{TP_{ii} + \sum_{j}^{n} F_{ij}}; \quad (2.28)$$

$$Recall_j = \frac{TP_{ii}}{TP_{ii} + \sum_{i}^{n} F_{ij}}; \quad (2.29)$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \quad (2.30)$$

где TP_{ii} - число верно классифицированных потоков;

F_{ij} – число неверно классифицированных потоков;

i – номер предсказанного класса;

j – номер истинного класса.

2.5 Методика отбора атрибутов

Для отбора рационального числа атрибутов на основе анализа их информативности можно зафиксировать допустимую вероятность ложной классификации и определить необходимое число атрибутов. Методика [106] включает в себя следующие шаги:

1) Задавшись вероятностью $FPR = const$, оценивается количество требуемых атрибутов $k_{\text{а}}$.
2) По найденному значению $k_{\text{а}}$ оценивается вероятность правильной классификации TPR.
3) По значению $k_{\text{а}}$ оценивается требуемое время обучения $T_{\text{об}}$ и время тестирования $T_{\text{тест}}$.
4) По значению $k_{\text{а}}$ оцениваются достижимые показатели эффективности: $Precision$; $Recall$; $Accuracy$; $F - measure$.

В условиях полной априорной определенности результаты классификации на основе алгоритмов Logistic Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF) и Gradient Boosting (GB) подтвердили правильность предложенного отбора атрибутов. При использовании 20 из 23 атрибутов вероятность ложной классификации может быть снижена в 2,5 раза с 0,007 до 0,0028 при уменьшении вероятности правильной классификации с 0,998 до 0,995.

Установлено, что для обеспечения достаточно высокого качества классификации мобильных приложений, осуществляющих шифрование сетевого трафика, достаточно ограничиться 13 наиболее информативными атрибутами. При этом размер обучающей выборки алгоритма RF для обеспечения классификации с достоверностью более 90% не превышает 300 потоков. Для такого же качества классификации достаточно анализировать от 16 до 58 пакетов в потоке (в

зависимости от приложения). Дальнейшее увеличение числа пакетов в потоке не приводит к заметному улучшению показателей эффективности.

2.6 Выбор атрибутов классификации

При классификации мобильных приложений на основе анализа сетевого трафика, важное значение имеют данные, используемые для обучения модели классификации (например, экспериментально полученный набор потока сетевых данных), и корректный выбор атрибутов. Процесс отбора атрибутов является важным подготовительным этапом обучения модели классификации, позволяющий оптимизировать процесс его обучения на предоставляемом наборе данных. Результат этого этапа может положительно сказаться на вычислительной сложности классификации.

В тех случаях, когда набор данных содержит большое количество данных, отбор атрибутов является необходимым этапом машинного обучения (МО), позволяющим сократить количество вычислительных операций. В результате будет получен набор данных, атрибуты которого будут отражать необходимую и достаточную информацию об интересующем классе, что в свою очередь ускорит процесс обучения модели классификации.

Отбор атрибутов проведём на двух наборах данных. Первый набор данных включает потоки сетевого трафика приложений, которые не используют шифрование (приложения a_1-a_6) – по 5000 потоков на приложение. Второй набор данных включает потоки трафика приложений, которые используют шифрование (приложения a_7-a_{12}).

Для выполнения фильтрации атрибутов использовались алгоритмы отбора, предоставляемые программной библиотекой Weka:

- Best First, который использует следующие алгоритмы выбора: A Correlation-based Feature Selector (CFS) и Wrapper.
- Greedy Stepwise, который использует следующие алгоритмы выбора: A Correlation-based Feature Selector (CFS) и Wrapper.

- Ranker, использует следующие алгоритмы выбора: Principal Components Analysis (PCA) и Information Gain Attribute Ranking (InfoGain).

Фильтрующий метод Wrapper использует в работе алгоритм классификации. Для каждого рассмотренного алгоритма классификации (Naive Bayes; C4.5; Random forest; SVM) был проведён отдельный отбор атрибутов.

Все проведенные эксперименты показали свою независимость от типа шифрования данных и от используемого алгоритма поиска. В качестве параметра алгоритма Wrapper указывалась модель классификации, с помощью которой выполнялся отбор. Все модели классификации, за исключением SVM, выделили одинаковый набор атрибутов. Алгоритм выбора Wrapper, использующий для своей работы модель классификации SVM, не отобрал информативных атрибутов. Отобранные атрибуты и результаты ранжирования атрибутов влгоритмо InfoGain представлены в таблице 2.4.

Таблица 2.4 – Выделенные наборы атрибутов

	Алгоритм CFS	Алгоритм PCA	Алгоритм Wrapper	Алгоритм InfoGain	
				>0.8	>1
1	x_{22}	x_1	x_{10}	x_1	x_2
2	x_{23}	x_2	x_{14}	x_2	x_3
3		x_3	x_{15}	x_3	x_4
4		x_4	x_{19}	x_4	x_6
5		x_5	x_{22}	x_5	x_8
6		x_6	x_{23}	x_6	x_9
7		x_7		x_7	x_{10}
8		x_8		x_8	x_{11}
9		x_9		x_9	x_{12}
10		x_{10}		x_{10}	x_{16}
11		x_{11}		x_{11}	x_{18}
12		x_{12}		x_{12}	x_{22}
13				x_{15}	x_{23}
14				x_{16}	
15				x_{17}	
16				x_{18}	
17				x_{22}	
18				x_{23}	
19					
20					
21					
22					

	Алгоритм CFS	Алгоритм PCA	Алгоритм Wrapper	Алгоритм InfoGain	
				>0.8	>1
23					
Итого	**2**	**12**	**6**	**18**	**13**

2.7 Результаты классификации мобильных приложений для выбранных наборов атрибутов

В ходе исследования влияния алгоритма отбора атрибутов на качество классификации мобильных приложений на основе анализа сетевого трафика было обнаружено, что алгоритмы обучения качественно классифицировали на следующих наборах атрибутов.

«Набор атрибутов Wrapper». Данный набор атрибутов поспособствовал получению высоких оценок у алгоритма классификации Naive Bayes, не зависимых от типа шифрования данных [107-108]. А также у алгоритмов классификации таких, как C4.5 и Random Forest при проведении экспериментов с зашифрованным сетевым трафиком.

«Набор атрибутов CFS». Данный набор атрибутов был выбран алгоритмами классификациями C4.5 и Random Forest при проведении экспериментов с мобильными приложениями, не осуществляющими шифрование сетевого трафика.

«Полный набор атрибутов». Данный набор атрибутов поспособствовал получению высоких оценок у алгоритма классификации SVM, не зависимых от типа шифрования данных.

Результаты классификации для алгоритмов KNN и RF на полном наборе атрибутов представлены в приложении А.

2.8 Влияние наличия фонового трафика на качество классификации

Фоновым трафиком, при классификации, является сетевой трафик таких мобильных приложений, которые отсутствовали в обучающем наборе данных модели классификации, но присутствуют в тестовом наборе.

Так как модели классификации не были обучены для классификации таких мобильных приложений, они способны только лишь отнести его к одному или нескольким известным классам.

Соответственно при добавлении фонового трафика в тестирующий набор данных модели классификации, то точность классификации снижается по сравнению с тем, когда фоновый трафик отсутствовал в тестирующем наборе данных.

Для проведения экспериментов были созданы следующие наборы данных сетевого трафика мобильных приложений:

- 12 приложений без фонового трафика.
- 12 приложений с фоновым трафиком.

Содержимое наборов представлено в таблице 2.5.

Таблица 2.5 – Характеристики наборов данных

Название набора	Приложение	Количество потоков	Приложения в качестве фонового трафика	Количество потоков
12 приложений без фонового трафика	ISG	4900		
	MI_RU	5000		
	PKB	5000		
	SB	5000		
	HSN	5000		
	SP	5000		
	MK	5000		
	NTV	5000		
	WR	5000		
	GVL	5000		
	PZ_EPR	5000		
	F_RE	5000		
12 приложений с фоновым трафиком	ISG	4900	Яндекс браузер с Алисой	5000
	MI_RU	5000	BK	5000
	PKB	5000	BD	4900
	SB	5000	KMS	5000
	HSN	5000	GG_CM	3800
	SP	5000	4PDA	4900
	MK	5000		
	NTV	5000		
	WR	5000		

Название набора	Приложение	Количество потоков	Приложения в качестве фонового трафика	Количество потоков
	GVL	5000		
	PZ_EPR	5000		
	F_RE	5000		

В качестве атрибутов был выбран набор, выделенный алгоритмом Wrapper: $x_{10}, x_{14}, x_{15}, x_{19}, x_{22}, x_{23}$.

Для проведения эксперимента использовались алгоритмы классификации Random Forest; Naive Bayes; SVM; AdaBoost; C4.5.

В таблице 2.6 показаны сводные данные параметра Precision (Точность) по всем алгоритмам для каждого класса, для обоих наборов данных. Как видно из таблицы 2.6 лучшими моделями классификации по параметру Precision (Точность) оказались Random Forest и C4.5. Фоновый трафик в тестирующем наборе данных привёл к изменению значений параметра Precision (Точность) у данных моделей классификации на десятые доли.

Таблица 2.6 – Сводная таблица параметра Precision (Точность)

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
ISG	0,989	0,319	0,487	0,133	0,387	0,122	0	0	0,99	0,261
MI_RU	0,988	0,184	0,446	0,19	0,525	0,243	0	0	0,983	0,192
SP	0,987	0,345	0,416	0,124	0,539	0,156	0	0	0,992	0,441
SB	0,973	0,692	0,557	0,123	0,647	0,171	0	0	0,972	0,337
F_RE	0,965	0,272	0,529	0,182	0,405	0,149	0	0	0,966	0,287
HSN	0,987	0,364	0,343	0,129	0,652	0,254	0	0	0,973	0,27
PKB	0,978	0,383	0,485	0,147	0,56	0,169	0	0	0,963	0,413
WR	0,999	0,961	0,901	0,702	0,16	0,692	0,134	0	1	0,994
PZ_EPR	0,997	0,96	0,629	0,457	0,847	0,61	0	0,051	0,995	0,939
MK	1	0,492	0,492	0,183	0,69	0,305	0	0	0,998	0,652
GVL	0,995	0,741	0,903	0,628	0,917	0,68	0,2	0,099	0,991	0,916
NTV	0,998	0,434	0,591	0,35	0,692	0,438	0	0	0,993	0,753

В таблице 2.7**Ошибка! Источник ссылки не найден.** показаны сводные данные параметра Recall (Полнота) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.7 – Сводная таблица параметра Recall (Полнота)

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
ISG	0,994	0,99	0,073	0,075	0,328	0,336	0	0	0,992	0,983
MI_RU	0,979	0,984	0,642	0,714	0,736	0,767	0	0	0,975	0,984
SP	0,989	0,989	0,773	0,769	0,742	0,731	0	0	0,996	0,989
SB	0,989	0,98	0,094	0,097	0,314	0,325	0	0	0,972	0,97
F_RE	0,976	0,985	0,593	0,613	0,522	0,552	0	0	0,971	0,976
HSN	0,984	0,989	0,296	0,28	0,305	0,295	0	0	0,982	0,979
PKB	0,977	0,97	0,542	0,537	0,773	0,774	0	0	0,964	0,968
WR	1	1	0,885	0,896	0,951	0,949	0,99	0	1	1
PZ_EPR	0,982	0,99	0,845	0,854	0,824	0,84	0	0,986	0,984	0,99
MK	1	0,998	0,483	0,459	0,54	0,532	0	0	0,998	0,998
GVL	0,993	0,993	0,94	0,94	0,927	0,992	0,976	0,993	0,994	0,992
NTV	0,99	0,994	0,513	0,555	0,63	0,654	0	0	0,989	0,984

Как видно из таблицы 2.7 лучшими моделями классификации по параметру Recall (Полнота) оказались Random Forest и C4.5. Фоновый трафик в тестирующем наборе данных привёл к изменению значений параметра Recall (Полнота) у данных моделей классификации на тысячные доли.

В таблице 2.8 показаны сводные данные параметра F-Measure (F-мера) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.8 – Сводная таблица параметра F-Measure (F-мера)

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
ISG	0,992	0,483	0,128	0,096	0,355	0,179	0	0	0,991	0,412
MI_RU	0,983	0,31	0,526	0,3	0,613	0,369	0	0	0,979	0,322
SP	0,988	0,512	0,541	0,214	0,624	0,258	0	0	0,994	0,61
SB	0,981	0,811	0,161	0,108	0,423	0,224	0	0	0,972	0,501
F_RE	0,972	0,427	0,559	0,281	0,456	0,235	0	0	0,969	0,443
HSN	0,985	0,533	0,318	0,177	0,416	0,273	0	0	0,977	0,423
PKB	0,977	0,549	0,512	0,231	0,649	0,277	0	0	0,963	0,579
WR	0,999	0,98	0,893	0,787	0,933	0,8	0,334	0	1	0,997
PZ_EPR	0,989	0,975	0,721	0,596	0,835	0,707	0	0,18	0,989	0,964
MK	1	0,659	0,488	0,262	0,605	0,388	0	0	0,998	0,789
GVL	0,994	0,849	0,921	0,753	0,922	0,783	0,236	0,097	0,993	0,953
NTV	0,994	0,604	0,549	0,43	0,659	0,524	0	0	0,991	0,853

Как видно из таблицы 2.8 лучшими моделями классификации по параметру F-Measure (F-мера) оказались Random Forest и C4.5. Фоновый трафик в тестирующем наборе данных привёл к изменению значений параметра F-Measure (F-мера) у данных моделей классификации на десятые доли.

В таблице 2.9 показаны сводные данные параметра Accuracy (Достоверность) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.9 – Сводная таблица параметра Accuracy (Достоверность)

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
Accuracy	0,988	0,405	0,558	0,232	0,634	0,262	0,161	0,067	0,985	0,403

Как видно из таблицы 2.9 лучшими моделями классификации по параметру Accuracy (Достоверность) оказались Random Forest и C4.5. Фоновый трафик в тестирующем наборе данных привёл к изменению значений параметра Accuracy (Достоверность) у данных моделей классификации на десятые доли.

В таблице 2.10 представлено время обучения и тестирования рассматриваемых алгоритмов.

Таблица 2.10 – Сводная таблица времени обучения и тестирования

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
Обучение	54986	53908	703	531	94629	89644	1266	1250	6735	7266
Тестирование	6141	13407	18282	41783	6547	10688	781	1563	1468	3812

На основе таблиц 2.7–2.10 были построены усреднённые гистограммы по каждому алгоритму классификации, они представлены на рисунках 2.1 – 2.2.

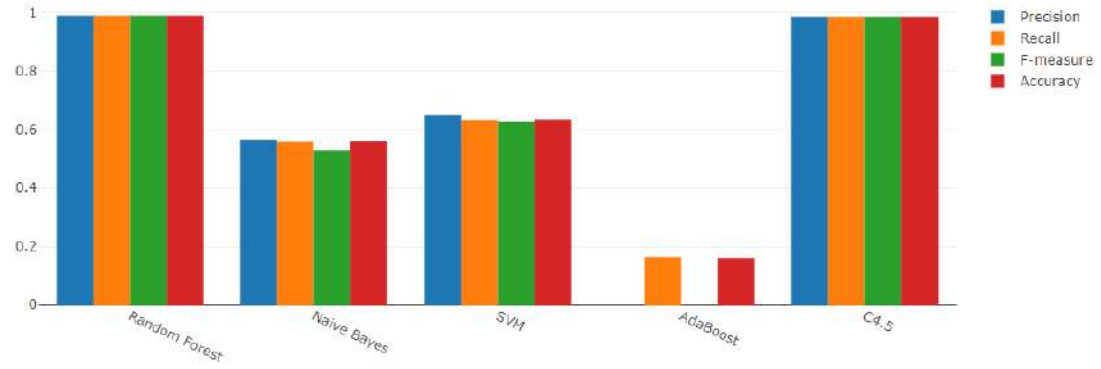


Рисунок 2.1 – Усреднённая гистограмма параметров качества классификации набора данных без фонового трафика

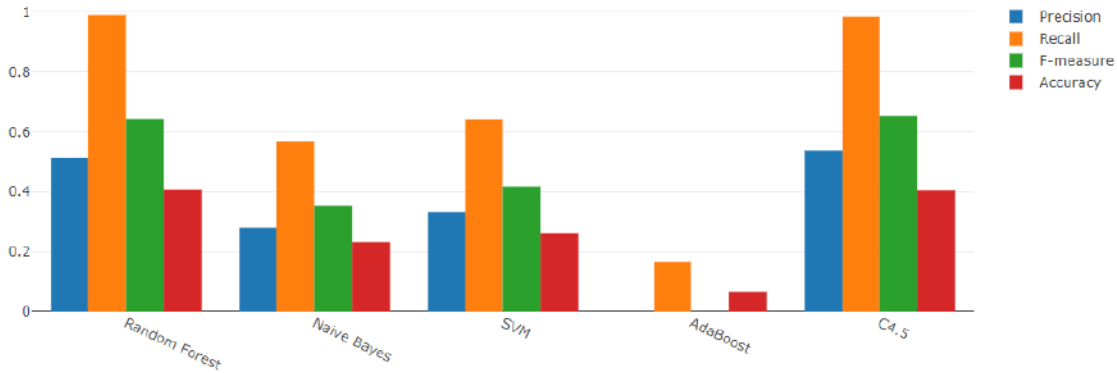


Рисунок 2.2 – Усреднённая гистограмма параметров качества классификации набора данных с фоновым трафиком

Исходя из полученных результатов можно сделать вывод, что лучшими из рассмотренных алгоритмов классификации являются Random Forest и C4.5, как для классификации без фонового трафика, так и в наборах с фоновым трафиком. В дальнейшем будут рассмотрены только эти алгоритмы классификации.

2.9 Классификация мобильных приложений на основе анализа сетевого трафика при наличии фонового трафика и класса «Неизвестное приложение»

Под априорной неопределённостью понимается наличие фонового трафика (ФТ), который не предусматривается в классических задачах ML при полной априорной информации о количестве и характеристиках классифицируемых приложений.

В [109] было показано, что отсутствие полной информации о структуре фонового трафика значительно снижает качество классификации приложений.

Существует два варианта решения этой проблемы: кластеризация сетевого трафика и введение в рассмотрение виртуального класса «Неизвестное приложение». Второй вариант является более простым и легко реализуемым.

Для проведения экспериментов были созданы следующие наборы данных сетевого трафика мобильных приложений:

- 5 приложений без фонового трафика.
- 5 приложений с фоновым трафиком.
- Различные типы приложений без фонового трафика;
- Различные типы приложений с фоновым трафиком.

Содержимое наборов представлено в таблице 2.11.

Таблица 2.11 – Характеристики наборов данных

Название набора	Приложение	Количество потоков	Приложения в качестве фонового трафика	Количество потоков
5 приложений без фонового трафика	SP	5000		
	SB	5000		
	HSN	5000		
	PKB	5000		
	MK	5000		
5 приложений с фоновым трафиком	SP	5000	BK	5000
	SB	5000	BD	4900
	HSN	5000	PZ_EPR	5000
	PKB	5000	YD_BS	5000
	MK	5000	NTV	5000
Различные типы приложений	MK	5000		
	SP	5000		
	YD_BS	5000		
	SB	5000		
	MI_RU	5000		
	HSN	5000		
	ISG	4900		
Различные типы приложений с фоновым трафиком	MK	5000		
	SP	5000		
	YD_BS	5000		
	SB	5000		
	MI_RU	5000		
	HSN	5000		

Название набора	Приложение	Количество потоков	Приложения в качестве фонового трафика	Количество потоков
	ISG	4900		

Для проведения эксперимента использовались алгоритмы классификации Random Forest; Naive Bayes; SVM; AdaBoost; C4.5.

В таблице 2.12 представлены сводные данные параметра Recall (Полнота) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.12 – Сводная таблица параметра Recall (Полнота)

Алгоритм	Random Forest			Naive Bayes			SVM			AdaBoost			C4.5		
Тип	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом
SP	0,962	0,983	0,954	0,64	0,766	0,767	0,767	0,751	0,734	0,937	0,937	0,934	0,974	0,993	0,932
SB	0,978	0,982	0,969	0,232	0,253	0,096	0,392	0,392	0,134	0	0	0	0,97	0,976	0,952
HSN	0,974	0,985	0,959	0,688	0,511	0,437	0,537	0,526	0,526	0	0	0	0,96	0,976	0,965
PKB	0,962	0,972	0,947	0,633	0,564	0,553	0,754	0,742	0,788	0	0	0	0,94	0,97	0,944
MK	0,996	0,999	0,996	0,576	0,59	0,382	0,834	0,84	0,624	0,996	0,996	0	0,996	0,997	0,998
Фон			0,99			0,924			0,967			0,952			0,993

В таблице 2.13 представлены сводные данные параметра Precision (Точность) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.13 – Сводная таблица параметра Precision (Точность)

Алгоритм	Random Forest			Naive Bayes			SVM			AdaBoost			C4.5		
Тип	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом
SP	0,972	0,461	0,934	0,6	0,239	0,318	0,592	0,284	0,461	0,266	0,118	0,151	0,979	0,241	0,925
SB	0,971	0,205	0,96	0,695	0,09	0,486	0,684	0,13	0,459	0	0	0	0,963	0,357	0,948

Алгоритм	Random Forest			Naive Bayes			SVM			AdaBoost			C4.5		
Тип	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом
HSN	0,966	0,478	0,955	0,434	0,185	0,383	0,746	0,243	0,473	0	0	0	0,945	0,636	0,938
PKB	0,965	0,345	0,951	0,535	0,267	0,497	0,587	0,328	0,459	0	0	0	0,957	0,32	0,942
MK	0,999	0,664	0,994	0,68	0,241	0,432	0,735	0,261	0,494	0,658	0,185	0	0,997	0,529	0,994
Фон			0,199			0,216			0,24			0,244			0,2

В таблице 2.14 показаны сводные данные параметра F-Measure (F-мера) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.14 – Сводная таблица параметра F-Measure (F-мера)

Алгоритм	Random Forest			Naive Bayes			SVM			AdaBoost			C4.5		
Тип	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом
SP	0,967	0,627	0,944	0,62	0,365	0,45	0,668	0,412	0,566	0,415	0,209	0,261	0,977	0,388	0,929
SB	0,974	0,339	0,964	0,348	0,133	0,161	0,499	0,195	0,207	0	0	0	0,966	0,522	0,95
HSN	0,97	0,644	0,957	0,532	0,272	0,408	0,625	0,332	0,498	0	0	0	0,953	0,77	0,951
PKB	0,964	0,509	0,949	0,58	0,362	0,523	0,66	0,455	0,58	0	0	0	0,948	0,481	0,943
MK	0,997	0,798	0,996	0,62	0,342	0,405	0,781	0,398	0,552	0,793	0,313	0	0,997	0,691	0,996
Фон			0,332			0,35			0,385			0,389			0,333

В таблице 2.15 показаны сводные данные параметра Accuracy (Достоверность) по всем алгоритмам для каждого класса, для обоих наборов данных.

Таблица 2.15 – Сводная таблица параметра Accuracy (Достоверность)

Random Forest			Naive Bayes			SVM			AdaBoost			C4.5		
Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом	Без фона	С фоном	С классом
0,961	0,37	0,583	0,438	0,201	0,316	0,539	0,244	0,377	0,187	0,145	0,187	0,953	0,369	0,58

В таблице 2.16 показаны время обучения и тестирования для всех алгоритмов классификации.

Таблица 2.16 – Сводная таблица времени обучения и тестирования

Алгоритм	Random Forest		Naive Bayes		SVM		AdaBoost		C4.5	
Тип набора данных	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном	Без фона	С фоном
Обучение	9094	17704	172	249	2531	6141	1046	1000	1078	1312
Тестирование	6172	12844	20064	36617	5235	12845	829	1156	2094	3532

Для наглядности сравнения на основе таблиц 2.13–2.16 были построены усреднённые гистограммы по алгоритмам, приведенные на рисунках 2.3–2.4.

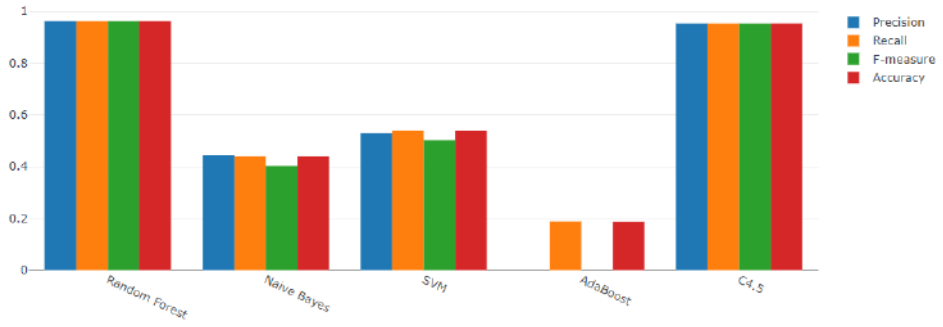


Рисунок 2.3 – Усреднённая гистограмма по алгоритмам классификации при отсутствии в наборе данных фонового трафика

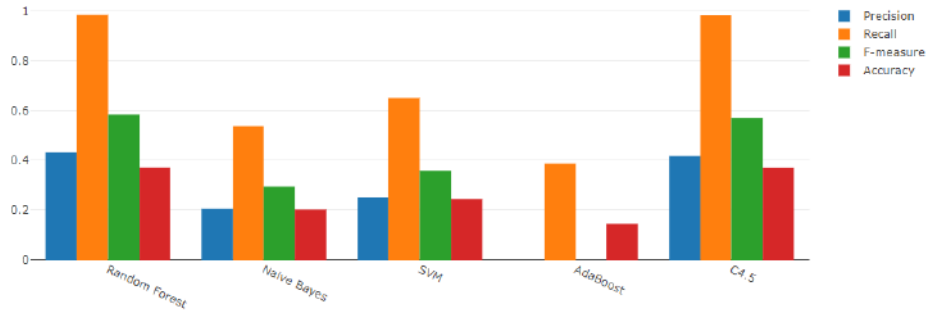


Рисунок 2.4 – Усреднённая гистограмма по алгоритмам классификации при наличии в наборе данных фонового трафика

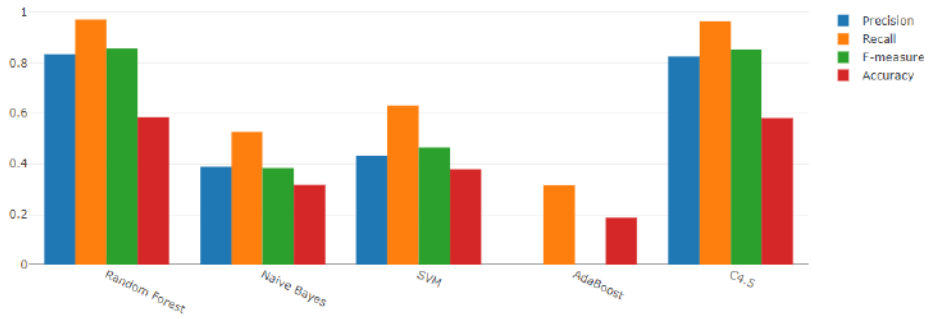


Рисунок 2.5 – Усреднённая гистограмма по алгоритмам классификации при наличии в наборе данных фонового трафика и наличии класса «Неизвестное приложение»

Из вышеописанных результатов следует, что класс «Неизвестное приложение» значительно увеличивает качество классификации с фоновым трафиком по сравнению с качеством классификации с фоновым трафиком, но без класса «Неизвестное приложение».

Однако он добавляет ложно положительных распределений по классам, что в свою очередь негативно сказывается на качестве классификации.

Однако в целом наличие класса «Неизвестное приложение» однозначно положительно сказывается на качество классификации при наличии фонового трафика.

Недостатком данного подхода является то, что класс «Неизвестное приложение» тоже необходимо размечать и использовать при обучении моделей классификации, что не всегда возможно, например, при появлении ранее неизвестного приложения.

2.10 Классификация мобильных приложений на основе анализа сетевого трафика при наличие фонового трафика с помощью автокодировщиков

2.10.1 Эффективность автокодировщиков

Эффективность АК достигается за счет фильтрации атрибутов перед классификацией. Для эксперимента были взяты приложения a_3 и a_9, осуществляющими шифрование сетевого трафиком. У обоих приложений были

рассмотрены 5 наиболее значимых атрибутов, без учета IP-адресов (атрибуты x_4, x_3, x_2, x_6, x_8). Столько же атрибутов наблюдалось на выходе. В качестве примера на рисунке 3 представлены гистограммы распределения пяти наиболее значимых атрибутов для всех приложений до фильтрации и с применением нормализации.

На рисунке 2.6 приведены гистограммы распределения атрибутов на входе и выходе АК. Видно, что специфика структуры АК приводит к уменьшению динамического диапазона изменения атрибутов на выходе АК. Число атрибутов на входе и выходе остается постоянным.

На гистограммах обозначено $mean = \frac{1}{N}\sum_{i=0}^{N-1} x_i$ – средне значение; $var = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \overline{x})^2$ – дисперсия; $std = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \overline{x})^2}$ – СКО распределения.

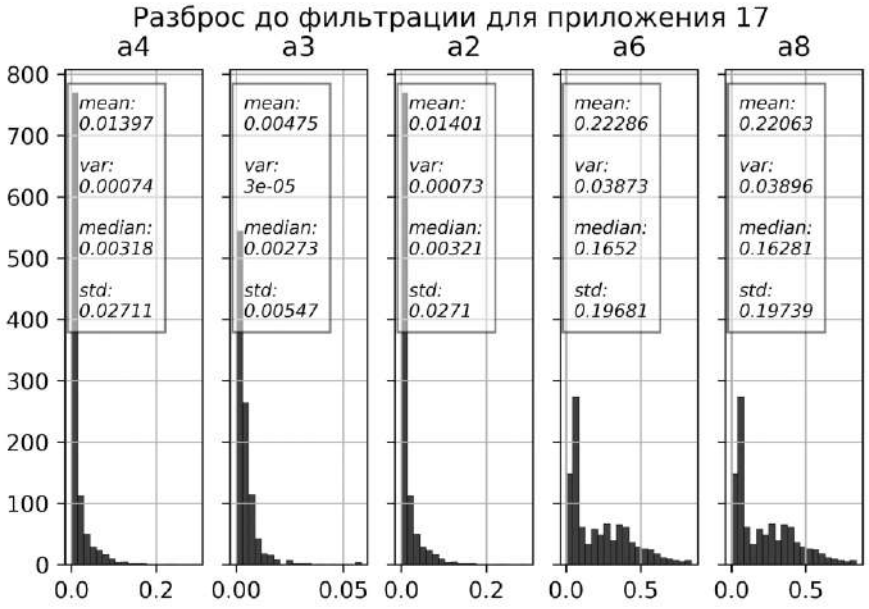
В таблице 2.18 приведены значения уменьшения СКО выходных данных выбранных атрибутов АК относительно входных (в процентах) для каждого приложения.

В таблице почти все диапазоны значений пяти наиболее значимых атрибутов для обоих приложений уменьшились до 2-х раз.

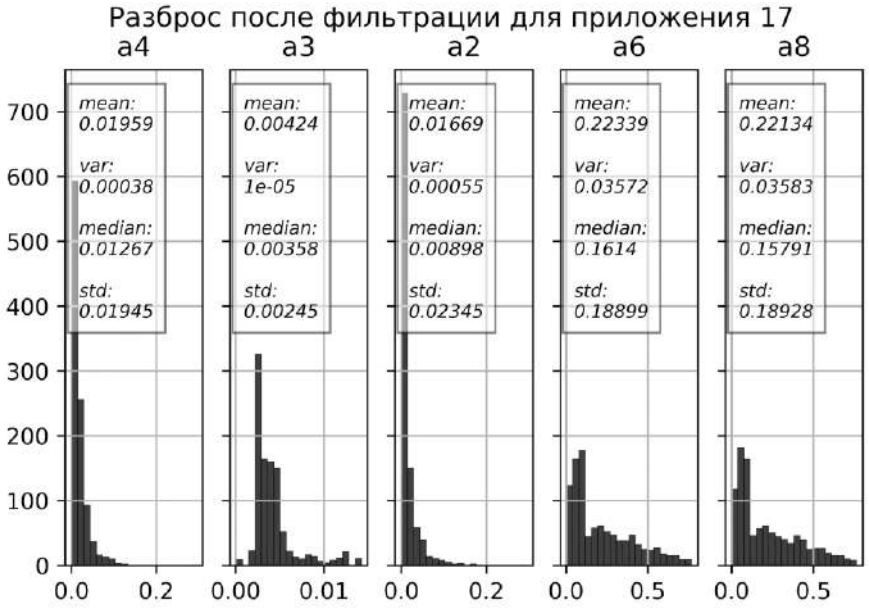
Таблица 2.17 – Уменьшение СКО на выходе автокодировщика в процентах

	x_4	x_3	x_2	x_6	x_8	Среднее значение
a_3	5.44	36.08	4.01	5.52	5.78	11,366
a_9	48.71	18.02	47.89	3.55	3.85	24.404
a_{10}	80.89	40.09	86.12	6.85	6.24	44,038
a_{12}	68.70	82.31	70.24	14.88	17.53	50,732
a_{14}	71.96	45.18	67.44	3.01	2.66	38,05
a_{17}	35.66	72.18	33.41	6.66	9.15	3 1,412

Как видно из таблицы, диапазоны значений пяти наиболее значимых атрибутов для анализируемых приложений уменьшаются от 11 до 50%.



а)



б)

Рисунок 2.6 – Гистограммы распределения атрибутов, иллюстрирующие разброс параметров а) на входе АК б) на выходе АК

2.10.2 Модель классификации с использованием АК

Автокодировщик может быть использован как часть модели бинарной классификации [110]. В процессе обучения АК обучается восстанавливать данные, т.е. изучать их структуру. В режиме предсказания АК будет восстанавливать

атрибуты приложения, на котором он обучался с минимальной ошибкой, в то время как при восстановлении атрибутов других приложений ошибка будет значительно больше.

Для того, чтобы определить уровень допустимой ошибки, часть обучающей выборки необходимо использовать для подбора порога ошибки. Эта часть выборки разделяется на группы по 32 экземпляра в каждой. В каждой группе рассчитывается пороговое значение, являющееся суммой выборочного среднего и трёх стандартных отклонений **MSE** (Mean Squared Error) или RMSE (Root Mean Squared Error). Усредненное пороговое значение по группам будем называть порогом экземпляра.

Если при восстановлении текущего вектора атрибутов среднеквадратическая ошибка превышает порог экземпляра, значит экземпляр не относится к данному классу. В противном случае считается, что экземпляр относится к данному классу.

Для того чтобы обобщить задачу бинарной классификации до многоклассовой потребуется построить ансамбль моделей бинарной классификации на основе автокодировщиков. Структура модели показана на рисунке 2.7.

В процессе обучения такой модели для каждого класса в обучающей выборке обучается отдельный АК и подбирается порог экземпляра. В процессе предсказания атрибуты экземпляра подаются на вход всех АК, вычисляется ошибка восстановления экземпляра и сравнивается с порогом экземпляра.

Предсказываемый класс выбирается среди тех АК чья ошибка не превысила порог путем выбора минимальное ошибки. В случае, если пороги превышены для всех АК, приложение считается фоновым или неизвестным.

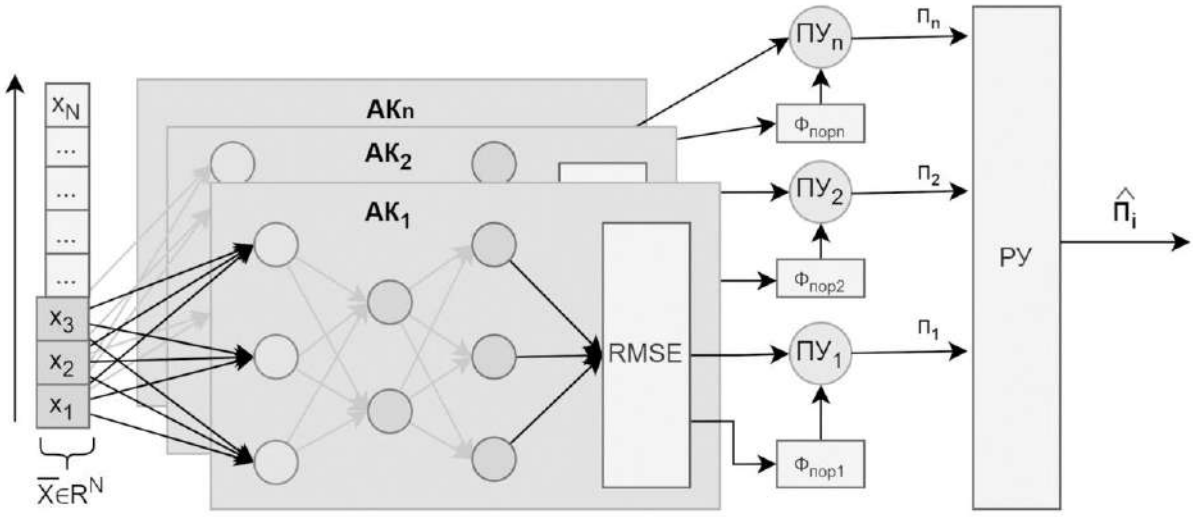


Рисунок 2.7 – Процедура классификации с помощью АК

2.10.3 Результаты классификации с помощью АК

При классификации противоправных, вредоносных и нежелательных мобильных приложений с использованием АК использовались значения гиперпараметров, представленные в таблице 2.19.

Таблица 2.18 – Параметры автокодировщика

Параметр	Значение
Тип автокодировщика	Vanilla Autoencoder
Количество слоёв	3
Количество скрытых слоёв	1
Размерность входного слоя	21
Размерность скрытого слоя	7
Функция активации скрытого слоя	Sigmoid
Функция активации выходного слоя	Linear
Оптимизатор	Adam
Размер группы	32

Результаты классификации с использованием АК представлены в таблице 2.20.

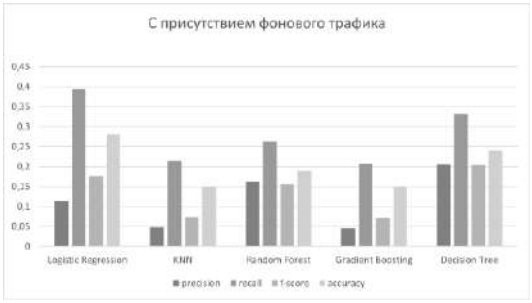
Таблица 2.19 – Метрики оценки результатов классификации с присутствием фонового трафика при использовании автокодировщиков

Приложение	Метрики			
	precision	recall	f-score	accuracy
a_3	1	0,98	0,99	0,97 (5%)

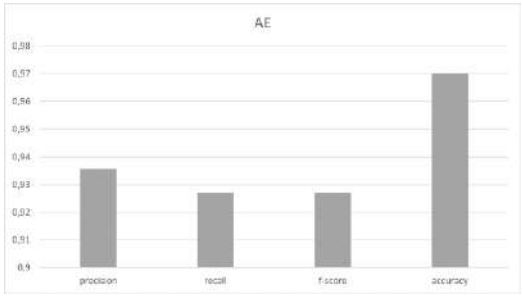
Приложение	Метрики			
	precision	recall	f-score	accuracy
a_9	1	0,88	0,93	
a_{10}	0,9	0,94	0,92	
a_{12}	0,95	0,8	0,87	
a_{14}	0,9	0,89	0,89	
a_{17}	0,8	1	0,89	
СРЕДНЕЕ	0,925	0,915	0,915	

Гистограммы оценки качества классификации с использованием АК, соответствующие данным в таблице 2.20, приведены на рисунке 2.8в.

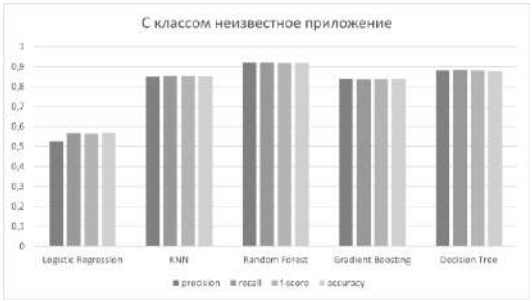
Сравнительный анализ полученных метрик в таблицах 3 и 6 показывает неоспоримое преимущество АК. Так сравнение с лучшим алгоритмом классификации RF показывает, что по достоверности (accuracy) выигрыш при использовании АК достигает 5%.



а)



б)



в)

Рисунок 2.8 – Усреднённые гистограммы по алгоритмам классификации а) с присутствием в наборе данных фонового трафика; б) при наличии в наборе данных фонового трафика и класса «Неизвестное приложение»; в) при наличии в наборе данных фонового трафика и обработке с автокодировщиком.

Для метрик precision, recall f-score выигрыш не превышает 2%. Однако несомненным преимуществом использования АК является отсутствие разметки фоновых приложений, особенно в случае их внезапного появления

Для повышения эффективности классификации в условиях фонового трафика предложено использовать ИНС – автокодировщик.

Автокодировщик – это модель глубокого обучения, которая обладает способностью изучать закодированное представление на нижнем слое, а затем воспроизводить входные данные на выходном уровне. ИНС состоят из L слоев нейронов, где каждый i-й слой $l^{(i)}$ последовательно соединен через синапсы со связанными весами $W^{(i)}$. Простой АК состоит из входного слоя, одного или нескольких скрытых и выходного слоя того же размера, что и входной. Эффективность АК достигается за счет фильтрации атрибутов перед классификацией.

Для эксперимента были взяты приложения a_3 и a_9, осуществляющих шифрование сетевого трафика. У обоих приложений рассмотрены 5 наиболее значимых атрибутов, без учета IP-адресов (атрибуты x_4, x_3, x_2, x_6, x_8).

В таблице 2.21 приведены значения уменьшения MSE (СКО) выходных данных выбранных атрибутов АК относительно входных (в процентах) для каждого приложения.

Таблица 2.20 – Уменьшение MSE (СКО) на выходе АК в процентах

	x_4	x_3	x_2	x_6	x_8	Среднее значение
a_3	5.44	36.08	4.01	5.52	5.78	11,366
a_9	48.71	18.02	47.89	3.55	3.85	24.404
a_{10}	80.89	40.09	86.12	6.85	6.24	44,038
a_{12}	68.70	82.31	70.24	14.88	17.53	50,732
a_{14}	71.96	45.18	67.44	3.01	2.66	38,05
a_{17}	35.66	72.18	33.41	6.66	9.15	3 1,412

Как видно из таблицы 2.21, диапазоны значений пяти наиболее значимых атрибутов для анализируемых приложений уменьшаются от 11 до 50%.

Автокодировщик может быть использован как часть модели бинарной классификации. В процессе обучения АК обучается восстанавливать данные, т.е. изучать их структуру. В режиме предсказания АК будет восстанавливать атрибуты приложения, на котором он обучался с минимальной ошибкой, в то время как при восстановлении атрибутов других приложений ошибка будет значительно больше.

Для того, чтобы определить уровень допустимой ошибки, часть обучающей выборки необходимо использовать для подбора порога ошибки. Эта часть выборки разделяется на группы по 32 экземпляра, в каждой из них рассчитывается пороговое значение, являющееся суммой выборочного среднего и трёх стандартных отклонений MSE (Mean Squared Error) или RMSE (Root Mean Squared Error). Усредненное пороговое значение по группам и есть порог экземпляра. Если при восстановлении текущего вектора атрибутов среднеквадратическая ошибка (MSE) превышает порог экземпляра, то экземпляр к данному классу не относится. В противном случае считаем, что экземпляр относится к данному классу.

Структура предложенной модели многоклассовой классификации на основе АК показана на рисунке 2.9 В процессе её обучения для каждого класса в обучающей выборке обучается отдельный АК и подбирается порог экземпляра. В процессе предсказания атрибуты экземпляра подаются на вход всех АК, вычисляется ошибка восстановления экземпляра и сравнивается с порогом экземпляра. Предсказываемый класс выбирается среди АК, чья ошибка не превысила порог и является минимальной. Если пороги превышены для всех АК, приложение считается фоновым или неизвестным.

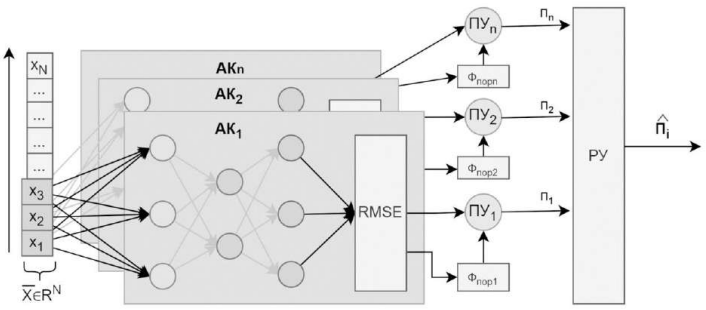


Рисунок 2.9 – Процедура классификации с помощью АК

При классификация противоправных, вредоносных и нежелательных мобильных приложений с использованием АК использовались значения гиперпараметров, представленные в таблице 2.22

Таблица 2.21 – Параметры автокодировщика

Тип автокодировщика	Vanilla Autoencoder
Количество слоёв	3 (Input, Bottleneck, Output)

Тип автокодировщика	Vanilla Autoencoder
Размерность входного слоя	Количество атрибутов в наборе данных за исключением целевого атрибута (21)
Размерность слоя Bottleneck	1/3 от размерности входного слоя (7)
Функция активации	Sigmoid
Оптимизатор	Adam
Функция потерь	MSE
Размер группы	32

Гистограммы оценки качества классификации с использованием АК в условиях фонового трафика приведены на рисунке 2.10в.

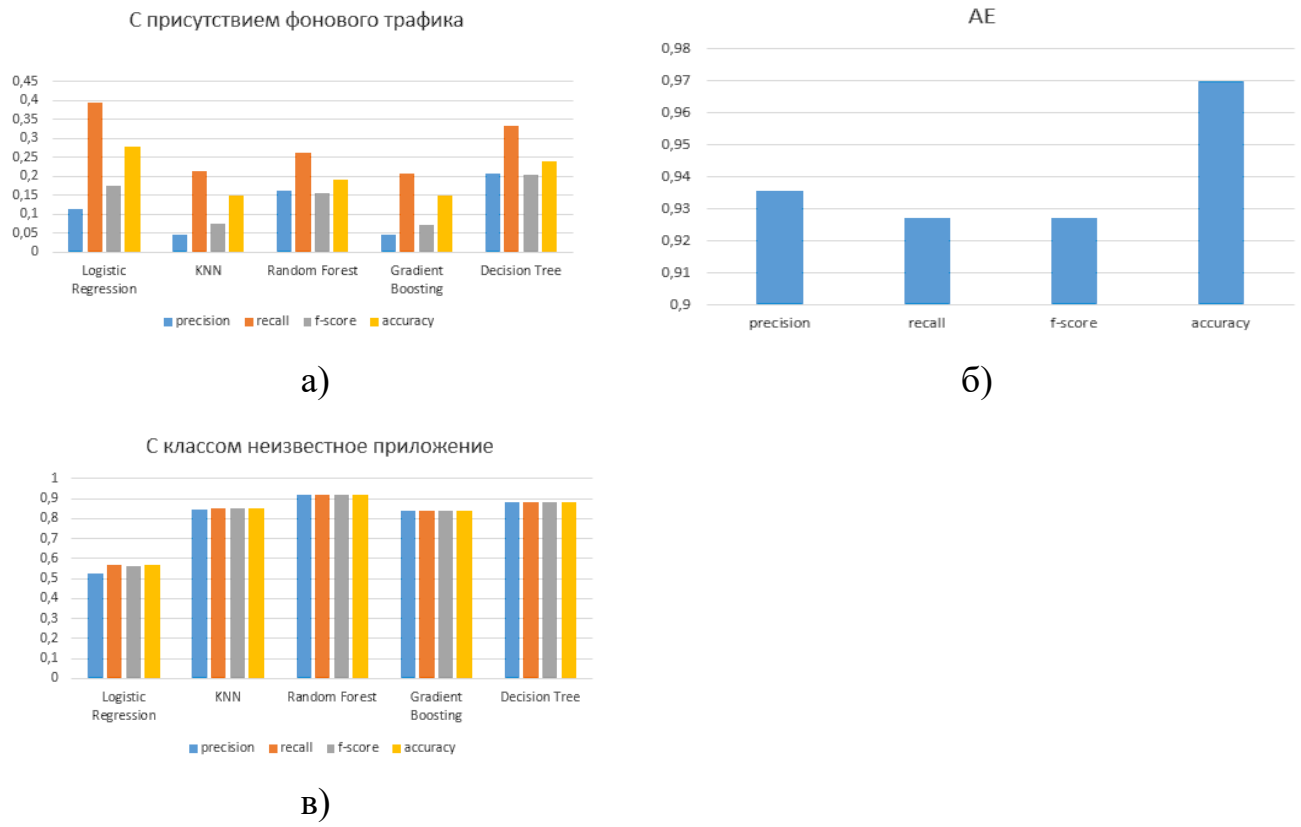


Рисунок 2.10 – Усреднённые гистограммы метрик алгоритмов классификации по всем классам а) при отсутствии фонового трафика; б) при наличии фонового трафика и класса «Неизвестное приложение»; в) при наличии фонового трафика и обработки с автокодировщиком

Анализ полученных метрик показывает неоспоримое преимущество АК. Сравнение с лучшим алгоритмом классификации RF даёт выигрыш по достоверности (accuracy) 5%. Для метрик precision, recall f-score выигрыш не превышает 2%. Однако преимуществом использования АК является отсутствие разметки фоновых приложений (особенно в случае их внезапного появления). Также достоинством АК является отсутствие дополнительных механизмов для

определения фонового трафика и повторного обучения в случае роста числа фоновых приложений.

Выводы

В ходе исследования влияния алгоритма выбора атрибутов на качество классификации мобильных приложений на основе анализа сетевого трафика было обнаружено, что алгоритмы обучения качественно классифицировали на следующих наборах атрибутов: “Набор атрибутов Wrapper”, “Набор атрибутов CFS”, “Полный набор атрибутов”.

«Набор атрибутов Wrapper» способствовал получению высоких оценок у алгоритма классификации Naive Bayes, не зависимо от типа шифрования данных, а также у алгоритмов классификации таких, как C4.5 и Random Forest при проведении экспериментов с шифрованным трафиком.

«Набор атрибутов CFS» был выбран алгоритмами классификациями C4.5 и Random Forest при проведении экспериментов с мобильными приложениями, не использующими шифрование сетевого трафика.

«Полный набор атрибутов» способствовал получению высоких оценок у алгоритма классификации SVM, не зависимо от типа шифрования данных.

В ходе классификации мобильных приложений, осуществляющих шифрование сетевого трафика с использованием алгоритма выбора атрибутов InfoGain [15] количество атрибутов было сокращено до 13.

Модель классификации AdaBoost является самой быстрой среди других исследуемых, но она так же имеет самые худшие результаты оценки классификации.

Модель классификации Random Forest наоборот является самой медленной, но имеет лучшие показатели оценки классификации.

Модели классификации Naive Bayes и SVM показали довольно средние результаты скорости и качества.

В ходе исследования влияния фонового трафика на качество классификации выявлено, что лучшими из рассмотренных алгоритмов классификации являются Random Forest и C4.5, как для классификации без фонового трафика, так и в наборах с фоновым трафиком. Дальнейшие исследования показали, что алгоритм C4.5 качественнее и быстрее работает на данных наборах чем Random Forest.

Класс «Неизвестное приложение» значительно увеличивает качество классификации с фоновым трафиком по сравнению с качеством классификации с фоновым трафиком, но без класса «Неизвестное приложение».

Однако он добавляет ложно положительных распределений по классам, что в свою очередь негативно сказывается на качестве классификации.

Однако в целом наличие класса «Неизвестное приложение» однозначно положительно сказывается на качество классификации при наличии фонового трафика.

Введение дополнительного класса «Неизвестное приложение» оказывает положительное воздействие на качество классификации с ФТ и позволяет повысить достоверность классификации в среднем на 20% при некотором увеличении числа ложно положительных решений. Недостатком такого подхода является необходимость размечать фоновый трафик и обучать модели классификации, что не всегда возможно, например, при появлении ранее неизвестного приложения.

Лучшим способом для классификации мобильных приложений при наличии фонового трафика является ансамбль нейронных сетей – автокодировщиков. Сравнение с лучшим алгоритмом классификации RF показывает, что наличие АК позволяет повысить достоверность классификации на 5%.

Основным достоинством АК является отсутствие дополнительных механизмов, для определения фонового трафика и повторного обучения, при росте числа фоновых приложений.

ГЛАВА 3 РАЗРАБОТКА МОДЕЛИ ОБНАРУЖЕНИЯ СМЕНЫ КОНЦЕПТА НА ОСНОВЕ АВТОКОДИРОВЩИКОВ

3.1 Обнаружение смены концепта с помощью автокодировщиков

В главе второй были получены наборы данных для обучения (2.25), валидации и тестирования (2.26). Воспользуемся ими для обучения и тестирования модели обнаружения смены концепта.

3.1.1 Обучение модели обнаружения смены концепта

Структурная схема модели обнаружения смены концепта [111] в режиме обучения представлена на рисунке 3.1.

Режим обучения предполагает наличия размеченных данных и заключается в построении АК для каждого приложения, подлежащего идентификации, и настройке порогов обнаружения смены концепта.

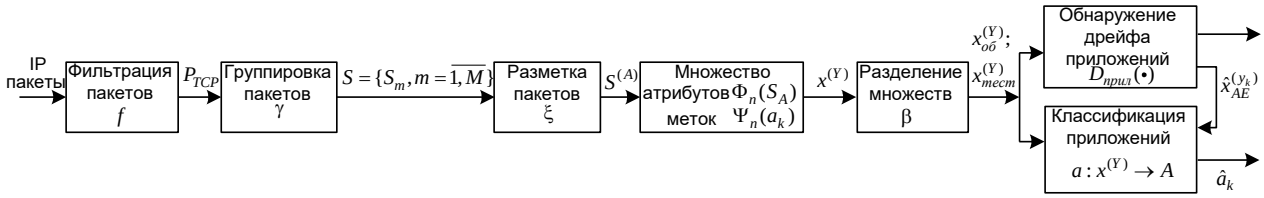


Рисунок 3.1 – Структурная схема алгоритма в режиме обучения

В структуре алгоритма, представленного на рисунке 3.1, предусмотрен алгоритм обнаружения смены концепта того или иного приложения TCP сеанса. Модель обнаружения смены концепта приложения в потоке $D_{прил}$ определяется через модель обнаружения смены концепта атрибута в потоке $D_{атриб}$ обученного на основе алгоритма автокодировщика $AE^{(y_k)}$. Множество реконструированных оценок на выходе автокодировщика $\hat{X}_{AE}^{(y_k)}$ позволяет принять решение о необходимости перестроить параметры алгоритма классификации $a: X_{об}^{(Y)} \to A$, при фиксации тренда или оставить их неизменными в случае отсутствия смены концепта.

3.1.2 Обнаружение смены концепта в потоке

Обнаружение смены концепта приложений в потоке представляет собой задачу многозначной классификации, т.к. возможны случаи, когда статистические характеристики изменяются сразу для нескольких приложений.

Обнаружение смены концепта в потоке осуществляется моделью D отдельно для каждой метки класса $y_k = \psi(a_k);\ k = \overline{1,K} | a_k \in A$. Для этого строятся модели $D^{(y_k)}$ обнаружения смены концепта в потоке с заданной меткой класса $y_k \in Y$.

Смена концепта считается обнаруженной, если хотя бы одна модель $D^{(y_k)}$ обнаружит её, т.е. обнаружена хотя бы в одном приложении. Разделение размеченного множества $X^{(Y)}$ на подмножества, содержащие только экземпляры с одинаковыми метками классов, осуществляется с помощью разделяющей функции β. Функция β получает на вход некоторое подмножество $X^{(Y)'} \subset X^{(Y)}$ и разделяет его на не пересекающиеся подмножества $X^{(Y)''}$:

$$\beta: \{X^{(Y)'} \subset X^{(Y)}\} \to \{\{X^{(y_k)'} \subset X^{(Y)'}\} \,|\, X^{(Y)'} \subset X^{(Y)}\};$$

$$\beta\left(X^{(Y)'}\right) = \{X^{(y_k)'} = \{\vec{x}_m; m = \overline{1, \left|X^{(Y)'}\right|} \,|\, (\vec{x}_m, y_k) \in X^{(Y)'}\}; k = \overline{1,K}\}\,.$$

При разделении множества на этапах обучения и тестирования в качестве меток классов используются истинные значения, в то время как на этапе предсказания – значения, предсказанные моделью классификации.

Обнаружение смены концепта приложения осуществляется моделью $D^{(y_k)}$ с помощью модели обнаружения смены концепта атрибутов приложения $D_{атриб}^{(y_k)}$.

Модель обнаружения смены концепта атрибутов $D_{атриб}^{(y_k)}$ основана на механизме обнаружения смены концепта в группах экземпляров $G^{(r)}$, которое осуществляется с помощью автокодировщика.

АК использует набор весовых коэффициентов распознавания для отображения входных данных в кодирующий вектор на скрытом слое, а затем использует набор генеративных весовых коэффициентов для восстановления закодированного вектора в исходный входной сигнал на выходном слое. Простой

АК состоит из входного слоя, одного или нескольких скрытых слоев и выходного слоя того же размера, что и входной слой [19,20,21].

Если АК обучен только на доброкачественных экземплярах, то он сможет реконструировать нормальные наблюдения, но не может реконструировать аномальные наблюдения (неизвестные понятия). В результате, когда АК фиксирует существенную ошибку восстановления, это классифицирует данные наблюдения как аномальные. Наличие смены концепта определяется с помощью оценок ошибок восстановления анализируемых приложений и превышения пороговых значений.

Этот подход заложен в представленном ниже алгоритмом обнаружения смены концепта для задачи многоклассовой классификации мобильных приложений с использованием АК.

3.1.3 Восстановление атрибутов приложений

Для каждого из анализируемых приложений $A = \{a_k; k = \overline{1,K}\}$, используя функции расчёта потерь при восстановлении, вычисляются три группы пороговых значений: пороги экземпляра приложения $k\,T_{экз}^{(k)}$, пороги группы приложения $k\,T_{гр}^{(k)}$ и пороги подсчета приложения $k\ T_n^{(k)}$. Группы пороговых значений состоят из пороговых значений атрибутов: **порог экземпляра** приложения k атрибута $n\ T_{экз}^{(kn)}$, **порог группы** приложения k атрибута $n\ T_{гр}^{(kn)}$ и **порог подсчета** приложения k атрибута $n\ T_n^{(kn)}$. Совокупность порогов атрибутов всех приложений будет выглядеть следующим образом:

$$T_{экз} = \{T_{экз}^{(k)} = \{T_{экз}^{(kn)}; k = \overline{1,K}\}; n = \overline{1,N}\},$$

$$T_{гр} = \{T_{гр}^{(k)} = \{T_{гр}^{(kn)}; k = \overline{1,K}\}; n = \overline{1,N}\},$$

$$T_n = \{T_n^{(k)} = \{T_n^{(kn)}; k = \overline{1,K}\}; n = \overline{1,N}\}.$$

Пороговое значение экземпляра $T_{экз}$ для нормальных (не подготовленных) данных вычисляется как среднее значение +3 (стандартное отклонение) значений ошибки восстановления. Предполагается, что любая точка данных со значениями

ошибок восстановления, превышающими пороговое значение экземпляра, будет смещенной точкой данных

Порог группы $T_{гр}$ вычисляется с использованием средних значений потерь при восстановлении пакета. Оно принимается как среднее значение +3 (стандартное отклонение) по сравнению со средними значениями ошибки восстановления пакета по всем данным проверки.

Порог подсчета $T_n^{(kn)}$приложения k атрибута n вычисляется с использованием средних значений потерь при восстановлении пакета. Оно принимается как среднее значение +3 (стандартное отклонение) по сравнению со средними значениями ошибки восстановления партии для всех партий по всем данным проверки.

Качество обученного автокодировщика $AE^{(y_k)}$ оценивается с помощью оценки качества восстановления экземпляров группы, вычисляемой с помощью функции Q на валидационной выборке $X_{AE_в}^{(y_k)}$. Для этого каждый экземпляр тестирующей выборки $X_{AE_в}^{(y_k)}$ подаётся на вход автокодировщику $AE^{(y_k)}$ и получается множество реконструированных экземпляров $\hat{X}_{AE_в}^{(y_k)} = AE^{(y_k)}(X_{AE_в}^{(y_k)})$.

Метрика качества $Q_в^{(y_k)} = Q(X_{AE_в}^{(y_k)}, \hat{X}_{AE_в}^{(y_k)}$ обученного автокодировщика $AE^{(y_k)}$ позволяет оценить качество восстановления экземпляров АК. При низком качестве хотя бы одного АК требуется его повторное обучение. После достижения достаточного качества восстановления можно перейти к расчёту пороговых значений.

В качестве функции потерь восстановления одного атрибута одного **экземпляра** воспользуемся функцией квадрата разности. Пусть $L_{атриб}$ – функция, вычисляющая потери восстановления n-ого атрибута $x_m^{(n)}$ экземпляра $\vec{x}_m$. Функция $L_{атриб}$ получает на вход исходный вектор атрибутов $\vec{x}_m$ из множества X, реконструированный вектор атрибутов $\hat{\vec{x}}_m$ из того же множества X, номер атрибута n, для которого осуществляется вычисление потерь восстановления, и возвращает

квадрат разности n-ых элементов исходного $\vec{x}_m$ и реконструированного $\hat{\vec{x}}_m$ векторов:

$L_{атриб}: \{\vec{x}_m \in X\}^2 \times \{n; n = \overline{1,N}\} \to \mathbb{R}$

$L_{атриб}(\vec{x}_m, \hat{\vec{x}}_m, n) = (x_m^{(n)} - \hat{x}_m^{(n)})^2, x_m^{(n)} \in \vec{x}_m, \hat{x}_m^{(n)} \in \hat{\vec{x}}_m$

Пусть $L_{атриб\ гр}: \{G^{(r)} \subset X\}^2 \times \{n; n = \overline{1,N}\} \to \mathbb{R}$ - функция потерь восстановления n-ого атрибута в группе экземпляров $G^{(r)}$, вычисляющая потери восстановления одного атрибута для всех экземпляров группы. Функция $L_{атриб\ гр}$ получает на вход исходную группу векторов атрибутов $G^{(r)} \subset X$, являющуюся подмножеством множества X, реконструировнную группу векторов атрибутов, $\hat{G}^{(r)} \subset X$, являющуюся подмножеством множества X, номер атрибута n, для которого осуществляется вычисление потерь, и возвращает множество, содержащее потери восстановления одного атрибута для всех экземпляров группы:

$L_{атриб}(G^{(r)}, \hat{G}^{(r)}, n) = \{L_{атриб}(\vec{x}_m, \hat{\vec{x}}_m, n); m = \overline{1, |G^{(r)}|}: \vec{x}_m \in G^{(r)}, \hat{\vec{x}}_m \in \hat{G}^{(r)}\}$

Пусть $L_{экз}: \{\vec{x}_m \in X\}^2 \to \mathbb{R}$ – функция вычисления потерь восстановления одного экземпляра, вычисляющая среднюю потерю восстановления всех атрибутов этого экземпляра. Функция $L_{экз}$ получает на вход исходный вектор атрибутов $\vec{x}_m$ из множества X, реконструированный вектор атрибутов $\hat{\vec{x}}_m$ из того же множества X и возвращает среднюю потерю восстановления всех атрибутов экземпляра (потерю восстановления экземпляра)

$L_{'экз}(\vec{x}_m, \hat{\vec{x}}_m) = \frac{1}{N} \sum_{n=1}^{N} L_{атриб}(\vec{x}_m, \hat{\vec{x}}_m, n).$

Функция качества восстановления <u>*одного атрибута экземпляров*</u> группы можно вычислить с помощью функции $Q_{атриб}: \{G^{(r)} \subset X\}^2 \times \{n; n = \overline{1,N}\} \to \mathbb{R}$ путём усреднения потери восстановления этого атрибута для всех экземпляров группы. Функция $Q_{атриб}$ получает на вход исходную группу векторов атрибутов $G^{(r)}$, которая является подмножеством множества X, реконструированную группу векторов атрибутов $\hat{G}^{(r)}$, являющуюся подмножеством множества X, номер атрибута n, для которого осуществляется вычисление потерь и возвращает среднюю потерю восстановления атрибута в группе:

$$Q_{атриб}(G^{(r)}, \hat{G}^{(r)}, n) = \frac{1}{|G^{(r)}|} \sum_{m=1}^{|G^{(r)}|} L_{атриб}(G_m^{(r)}, \hat{G}_m^{(r)}, n), G_m^{(r)} \in X, \hat{G}_m^{(r)} \in X$$

Функция качества восстановления экземпляров группы можно вычислить с помощью функции $Q: \{G^{(r)} \subset X\}^2 \to \mathbb{R}$ путём усреднения потерь восстановления одного экземпляра. Функция Q получает на вход исходную группу векторов атрибутов $G^{(r)}$, являющуюся подмножеством множества X, реконструированную группу векторов атрибутов $\hat{G}^{(r)}$, являющуюся подмножеством множества X, возвращает среднее значение потерь восстановления экземпляров:

$$Q(G^{(r)}, \hat{G}^{(r)}) = \frac{1}{|G^{(r)}|} \sum_{m=1}^{|G^{(r)}|} L_{экз}(G_m^{(r)}, \hat{G}^{(r)}), G_m^{(r)} \in X, \hat{G}^{(r)} \in X$$

Пример вычисления ошибок восстановления экземпляров в группе представлен в таблице 3.1

Таблица 3.1 – Вычисление ошибок восстановления экземпляров в группе

$G^{(r)}$	$\hat{G}^{(r)}$	1	…	N	Результат
$\vec{x}_1$	$\hat{\vec{x}}_1$	$L_{\text{атриб}}(\vec{x}_1, \hat{\vec{x}}_1, 1)$	…	$L_{\text{атриб}}(\vec{x}_1, \hat{\vec{x}}_1, N)$	$L_{экз}(\vec{x}_1, \hat{\vec{x}}_1)$
…	…	…	…	…	…
$\vec{x}_{\lvert G^{(r)}\rvert}$	$\hat{\vec{x}}_{\lvert G^{(r)}\rvert}$	$L_{\text{атриб}}(\vec{x}_{\lvert G^{(r)}\rvert}, \hat{\vec{x}}_{\lvert G^{(r)}\rvert}, 1)$	…	$L_{\text{атриб}}(\vec{x}_{\lvert G^{(r)}\rvert}, \hat{\vec{x}}_{\lvert G^{(r)}\rvert}, N)$	$L_{\text{экз}}(\vec{x}_{\lvert G^{(r)}\rvert}, \hat{\vec{x}}_{\lvert G^{(r)}\rvert})$
		$Q_{\text{атриб}}(G^{(r)}, \hat{G}^{(r)}, 1)$	…	$Q_{\text{атриб}}(G^{(r)}, \hat{G}^{(r)}, N)$	$Q(G^{(r)}, \hat{G}^{(r)})$

Приведённые функции потерь и функции качества используются для оценки качества обученных АК и вычисления порогов.

3.1.4 Алгоритм обнаружения смены концепта

Процесс обучения модели обнаружения смены концепта в атрибутах включает обучение АК, расчёт потерь восстановления атрибутов и расчёт пороговых значений.

Разделим с помощью описанной выше разделяющей функции β обучающую выборку $X_{об}^{(Y)}$ на k компонент $X_{об}^{(Y)} = U_{X_{об}^{(y_k)} \in \beta(X_{об}^{(Y)})} X_{об}^{(y_k)}$.

Каждое полученное множество $X_{об}^{(y_k)}$ содержит только векторы атрибутов, которые помечены меткой класса y_k, то есть представляет собой TCP сеансы, которые принадлежат приложению a_k.

Разобьем каждое множество $X_{об}^{(y_k)}$ на 3 непересекающихся множества: обучающее $X_{AE_{об}}^{(y_k)}$, валидационное $X_{AE_в}^{(y_k)}$ множества и множество для настройки пороговых значений $X_{AE_п}^{(y_k)}$, так что $X_{об}^{(y_p)} = X_{AE_{об}}^{(y_p)} \cup X_{AE_в}^{(y_p)} \cup X_{AE_п}^{(y_p)}$; $X_{AE_{об}}^{(y_p)} \cap X_{AE_в}^{(y_p)} = \emptyset$; $X_{AE_{об}}^{(y_p)} \cap X_{AE_п}^{(y_p)} = \emptyset$; $X_{AE_в}^{(y_p)} \cap X_{AE_п}^{(y_p)} = \emptyset$.

Будем считать, что обучающая выборка составляет 80% от исходной обучающей выборки $\left|X_{AE_{об}}^{(y_k)}\right| = 0.8\left|X_{об}^{(y_k)}\right|$, валидационная выборка $\left|X_{AE_в}^{(y_k)}\right| = 0.05\left|X_{об}^{(y_k)}\right|$– 5%, а выборка для настройки пороговых значений $\left|X_{AE_п}^{(y_k)}\right| = 0.15\left|X_{об}^{(y_k)}\right|$– 15%.

Полученные выборки будут использоваться для обучения, валидации и настройки пороговых значений.

Для обнаружения изменения в данных воспользуемся моделью автокодировщика $AE: \{X_{вх} \subset X\} \to \{X_{вых} \subset X\}$, обучаемой с помощью алгоритма $\lambda: \{X' \big| X' \subset X\} \to AE\}$.

На каждой обучающей выборке $X_{AE_{об}}^{(y_k)}$ автокодировщик $AE^{(y_k)}$ обучается, что может быть описано в виде $AE^{(y_k)} = \lambda(X_{AE_{об}}^{(y_k)})$.

Введём в рассмотрение индикаторную функцию$I(x) = \begin{cases} 1, \textit{если } x \textit{ истино} \\ 0, \textit{в противном случае} \end{cases}$.

Тогда модель обнаружения смены концепта атрибута в группе $D_{атриб}^{гр}$ может быть записана в виде:

$$D_{атриб}^{гр}(G^{(r)}, \hat{G}^{(r)}, n, T_{гр}^{(k)}, T_{экз}^{(k)}, T_п^{(k)}) = I(Q_{атриб}(G^{(r)}, \hat{G}^{(r)}, n) > T_{гр}^{(kn)}) I(T_{п\ атриб\ гр}(G^{(r)}, \hat{G}^{(r)}, n, T_{экз}^{(kn)}) > T_п^{(kn)}),$$

где $G^{(r)}$– группа экземпляров, для которой обнаруживается смена концепта атрибута, $\hat{G}^{(r)}$ – группа реконструированных с помощью автокодировщика экземпляров, n – номер атрибута, для которого обнаруживается смена концепта, $T_{гр}^{(k)}$– множество порогов группы для всех атрибутов приложения k, $T_{экз}^{(k)}$– множество порогов экземпляров для всех атрибутов приложения k,$T_n^{(k)}$ – множество порогов подсчёта для всех атрибутов приложения k, $T_{гр}^{(kn)} \in T_{гр}^{(k)}$– порог группы для n-ого атрибута приложения k, $T_{экз}^{(kn)} \in T_{экз}^{(k)}$ – порог экземпляра для n-ого атрибута приложения k, $T_n^{(kn)} \in T_n^{(k)}$ – порог подсчёта для n-ого атрибута приложения k, $T_{n\ \text{атриб гр}}$ – функция вычисления количества экземпляров группы $G^{(r)}$, для которых потеря восстановления превышает порог экземпляров $T_{экз}^{(kn)}$.

Группа экземпляров считается моделью $D_{атриб}^{гр}$ дрейфующей, если средние потери восстановления группы $G^{(r)}$, вычисленные с помощью функции $Q_{атриб}$, превышают порог группы $T_{гр}^{(kn)}$, а количество экземпляров группы $G^{(r)}$, для которых потеря восстановления превышает порог экземпляров $T_{экз}^{(kn)}$, превышает порог подсчёта $T_n^{(kn)}$.

Модель обнаружения смены концепта атрибута в потоке $D_{атриб}$ определяется через модель обнаружения смены концепта атрибута в группе $D_{атриб}^{гр}$:

$$D_{атриб}(G, \hat{G}, n, T_{гр}^{(k)}, T_{экз}^{(k)}, T_n^{(k)}, w) =$$

$$\begin{cases} r_{дрейф}, \text{если } \exists r_{дрейф} > 0 : \left[\prod_{r=r_{дрейф}}^{r_{дрейф}+w} D_{атриб}^{гр}(G^{(r)}, \hat{G}^{(r)}, n, T_{гр}^{(k)}, T_{экз}^{(k)}, T_n^{(k)})\right] > 0, \\ 0, \text{в противном случае} \end{cases}$$

где G– исходное множество групп экземпляров, $\hat{G}$ – множество реконструированных групп экземпляров, n – номер атрибута, для которого обнаруживается смена концепта, $T_{гр}^{(k)}$ – множество порогов группы для всех атрибутов, $T_{экз}^{(k)}$ – множество порогов экземпляров для всех атрибутов, $T_n^{(k)}$– множество порогов подсчёта для всех атрибутов, w – количество подряд идущих групп, в которых должна определиться смена концепта.

Атрибут n считается дрейфующим в потоке, если в множестве экземпляров G обнаруживается w подряд идущих групп, для которых модель $D_{атриб}^{гр}$в атрибуте n обнаруживает смену концепта. Модель возвращает номер группы, с которой началась смена концепта, либо 0, в противном случае.

Модель обнаружения смены концепта приложения в потоке $D_{прил}$ определяется через модель обнаружения смены концепта атрибута в потоке $D_{атриб}$:

$$D_{прил}(G, \hat{G}, T_{гр}^{(k)}, T_{экз}^{(k)}, T_{n}^{(k)}, w) = \{(n, r) | r = D_{атриб}(G, \hat{G}, n, T_{гр}^{(k)}, T_{экз}^{(k)}, T_{n}^{(k)}, w), r > 0, n = \overline{1, N}\}$$

где G – исходное множество групп экземпляров, $\hat{G}$ – множество реконструированных групп экземпляров, $T_{гр}^{(k)}$– множество порогов группы для всех атрибутов, $T_{экз}^{(k)}$– множество порогов экземпляров для всех атрибутов, $T_{n}^{(k)}$– множество порогов подсчёта для всех атрибутов, w – количество подряд идущих групп, в которых должна определиться смена концепта.

Для каждого атрибута с помощью модели обнаружения смены концепта атрибута $D_{атриб}$ определяется смена концепта атрибута. Смена концепта приложения обнаруживается, если она обнаруживается хотя бы в одном атрибуте этого приложения. Модель обнаружения смены концепта приложения $D_{прил}$ возвращает множество пар – номер атрибута n, в котором обнаружена смена концепта, и номер группы r, с которой она началась. Если в атрибуте n смена концепта не обнаружена ($r = 0$), то он не попадает в результирующее множество. В случае если смена концепта не обнаружена ни в одном атрибуте, результирующее множество будет пустым. Итоговая модель обнаружения смены концепта будет иметь вид:

$$D(X^{(Y)'}, w) = \{(y_k, n, r) | (n, r) \in D_{прил}(G^{(y_k)}, AE^{(y_k)}(G^{(y_k)}), T_{гр}^{(k)}, T_{экз}^{(k)}, T_{n}^{(k)}, w), G^{(y_k)} \in g(X^{(y_k)}, V), X^{(y_k)} \in \beta(X^{(Y)'})\}$$

где $X^{(Y)'}$ - размеченное множество TCP сеансов, представленное в виде множества пары вектора атрибутов $\vec{x}_m \in X$ и метки класса $y_m^{'} \in Y$, w– количество подряд идущих групп, в которых должна определиться смена концепта, $X^{(y_k)}$–

размеченное подмножество TCP сеансов, сгенерированное приложением с меткой класса y_k.

Обнаружение смены концепта может осуществляться не только на этапах обучения и тестирования, но и на этапе предсказания, т.к. не предполагает наличия истинных меток и заключается либо в использовании меток модели классификации, либо в попытке восстановления данных с использованием всех АК путём подсчёта ошибок восстановления. Структурная схема предлагаемой МОСК, основанной на механизме обнаружения смены концепта в группах экземпляров, приведена на рисунке 3.2. При обучении множество АК используют обычные, размеченные данные.

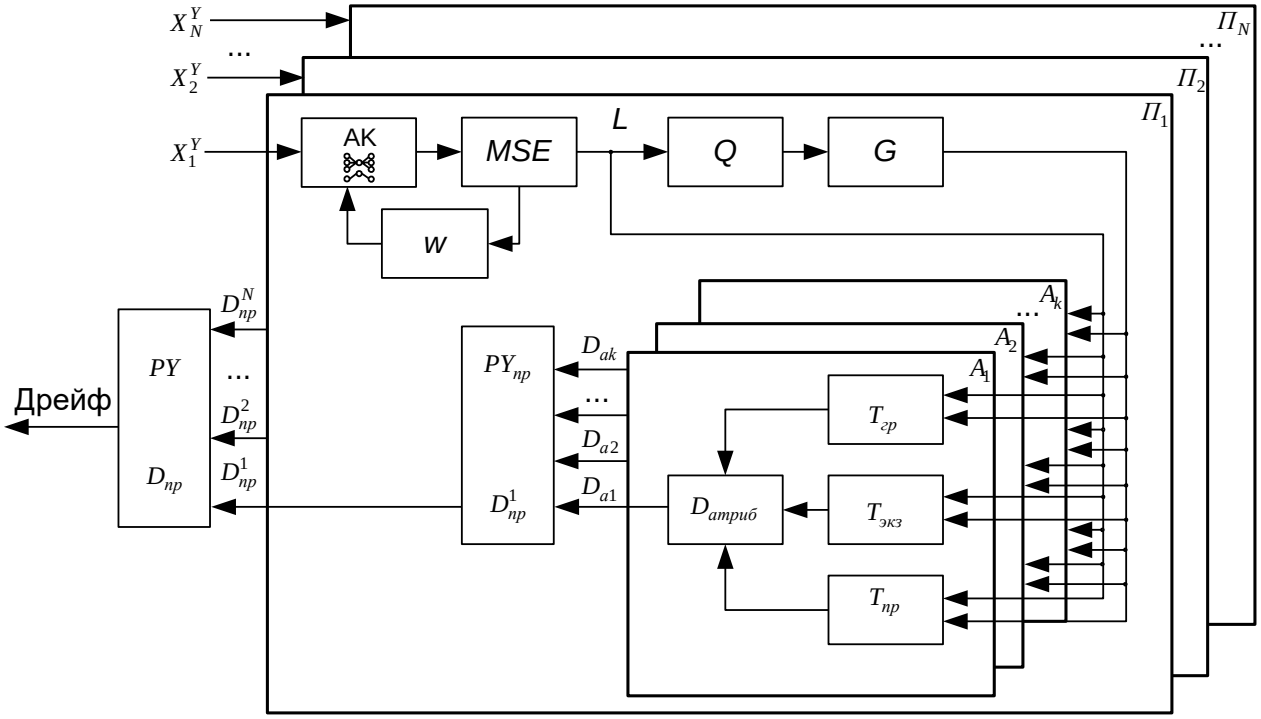


Рисунок 3.2 – Структурная схема МОСК

На этапе обучения множество АК обучаются на обычных размеченных данных.

Задача обнаружения смены концепта заключается в определении приложений, статистические характеристики атрибутов которых значительно изменились и качество классификации которых заметно снизилось.

Автокодировщик может быть использован как часть модели бинарной классификации. В процессе обучения АК обучается восстанавливать данные, т.е.

изучать их структуру. В режиме предсказания АК будет восстанавливать атрибуты приложения, на котором он обучался с минимальной ошибкой, в то время как при восстановлении атрибутов других приложений ошибка будет значительно больше.

Обучающее множество $X_{об}^{(Y)}$ разделяется на k подмножеств, равное количеству идентифицируемых приложений. Каждое подмножество $X_{об}^{(y_k)}$ в свою очередь разделяется на обучающее $X_{AE_{об}}^{(y_k)}$, валидационное $X_{AE_в}^{(y_k)}$ множества и множество для настройки порогов приложения $X_{AE_п}^{(y_k)}$. Обучающее множество приложения $X_{AE_{об}}^{(y_k)}$ используется для обучения автокодировщика, а валидационное $X_{AE_в}^{(y_k)}$ – для его оценки.

Множество для настройки порогов $X_{AE_п}^{(y_k)}$ приложения разделяется на группы экземпляров одинакового размера и используется для вычисления порогов экземпляра $T_{экз}^{(k)}$, группы $T_{гр}^{(k)}$ и подсчета $T_п^{(k)}$.

Каждый экземпляр в группе реконструируется (восстанавливается) с помощью обученного АК. Для каждого экземпляра в группах рассчитывается функция потерь. В качестве функции потерь была выбрана среднеквадратичная ошибка $MSE = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \hat{x}_i)^2$. Потери атрибутов в каждой группе усредняются и получаются средние потери атрибута в группе. Усреднив потери атрибутов по группам, получим пороги группы для каждого атрибута $T_{гр}^{(k)} = \{T_{гр}^{(kn)}; n = \overline{1,N}\}$. В каждой группе для каждого атрибута также вычисляется 0,99-квантиль ошибки (значение, которое заданная случайная величина атрибута не превышает с фиксированной вероятностью 0,99). Усреднив по первым нескольким группам ($G^{(r)}; r = \overline{1,R_{усp}}, R_{усp} < R$) , получим пороги экземпляра атрибутов $T_{экз}^{(k)} = \{T_{экз}^{(kn)}; n = \overline{1,N}\}$. В каждой группе вычисляется количество экземпляров, превышающее порог экземпляра. Максимальное значение среди групп будут являться порогами подсчёта атрибутов $T_п^{(k)} = \{T_п^{(kn)}; n = \overline{1,N}\}$.

Если при восстановлении текущего вектора атрибутов среднеквадратическая ошибка превышает порог экземпляра, значит экземпляр не относится к данному классу. В противном случае считается, что экземпляр относится к данному классу.

В процессе обучения такой модели для каждого класса в обучающей выборке обучается отдельный АК и подбирается порог экземпляра. В процессе предсказания атрибуты экземпляра подаются на вход всех АК, вычисляется ошибка восстановления экземпляра и сравнивается с порогом экземпляра.

Предсказываемый класс выбирается среди тех АК чья ошибка не превысила порог путем выбора минимальное ошибки. В случае, если пороги превышены для всех АК, приложение считается фоновым или неизвестным.

Реализация разработанного алгоритма обнаружения смены концепта на примере описанного выше набора данных была реализована в рамках программной среды Python. Значения параметров анализируемого трафика приведены в табл. 3.2

Таблица 3.2 – Значения параметров потока анализируемого трафика

Параметр	Описание параметра	Значение
M	Общее количество TCP сеансов	30 000
K	Общее количество приложений	6
N	Общее количество атрибутов	21

Для каждого приложения обучаются автокодировщики количество которых равно количеству анализируемых приложений.

Параметры автокодировщиков, реализованных в рамках разработанного алгоритма обнаружения смены концепта представлены в таблице 3.3.

Таблица 3.3 – Параметры автокодировщика

Параметр	Значение
Тип автокодировщика	Vanilla Autoencoder
Количество слоёв	3
Количество скрытых слоёв	1
Размерность входного слоя	21
Размерность скрытого слоя	7
Функция активации скрытого слоя	Sigmoid
Функция активации выходного слоя	Linear
Оптимизатор	Adam
Размер группы	32

Описанный алгоритм обнаружения дрейфа реализован на языке программирования Python. Исходный код представлен в приложении Б.

3.2 Обнаружение смены концепта с учётом эффекта старения данных

Для обнаружения смены концепта в потоковом режиме предложено определять значение текущих статистических характеристик атрибутов трафика мобильных приложений с помощью двух перемещающихся во времени окон с учетом эффекта старения поступающих данных [112]. Разработанный алгоритм обнаружения смены концепта в потоке данных, базируется на этом механизме. Текущий анализ приложений осуществляется с помощью двух скользящих окон W_1 и W_2, контролируя изменение текущих статистических характеристик атрибутов мобильных приложений.

Предложенное решающее правило для обнаружения смены конецепта может быть записано в виде неравенства:

$$R_t^J = \frac{M_{W_2}^J(t)}{M_{W_1}^J(t)} > \lambda$$

где $M_{W_1}^J(t) = \frac{\frac{1}{K}\sum_{i=N2}^{N1+N2-1}\sum_{j=1}^{K}\alpha_1^i y_{t-i,j}^J}{\sum_{i=0}^{N1-1}\alpha_1^i}$,

$M_{W_2}^J(t) = \frac{\frac{1}{K}\sum_{i=0}^{N2-1}\sum_{j=1}^{K}\alpha_2^i y_{t-i,j}^J}{\sum_{i=0}^{N2-1}\alpha_2^i}$

y_t – значение элементов наблюдаемого потока $Y = \{y_0, y_1, \dots, y_{N-1}\}$ (приложений и атрибутов), измеренных в момент времени $t \in T = \{0,1, \dots, N-1\}$;

N – размер множества Y;

α_1, α_2 – коэффициенты затухания, характеризующие «память» измеренных значений соответственно в первом W_1 и втором W_2 окнах ($0 \ll \alpha_2 < \alpha_1 < 1$).

В вычислительных экспериментах использовались значения параметров: $N_1 = 50000$; $N_2 = 10000$; $\alpha_1 = 0.999$; $\alpha_2 = 0.997$. Результаты показали, что выбор коэффициентов затухания α_1 и α_2, размеров окон W_1 и W_2 достаточно критичен, требует «ручного» задания пользователем. При этом обнаружение смены концепта происходит несколько быстрее.

На рисунке 4 представлены зависимости вероятности ошибок классификации от объема поступающих данных.

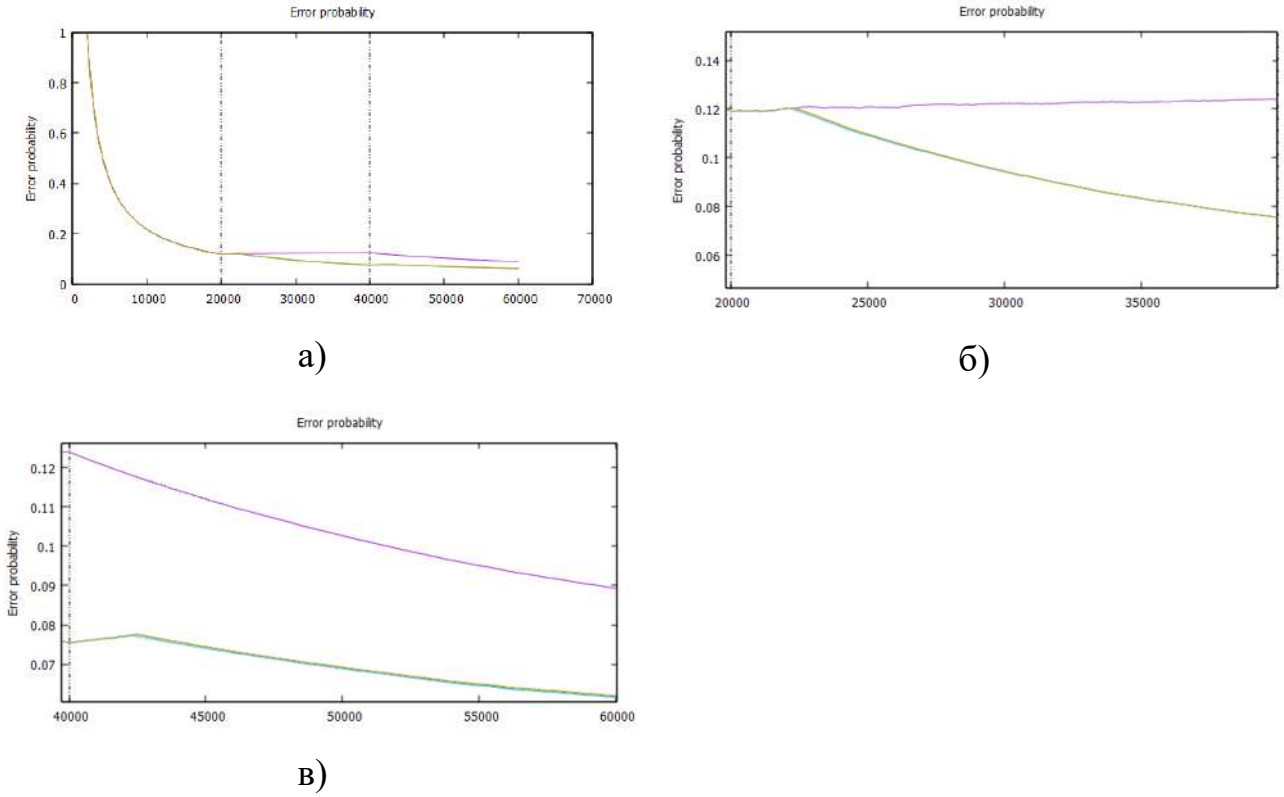


Рисунок 3.3 – Зависимости вероятности ошибок от объёма поступающих данных а) до смены концепта б) после первой смены концепта в) после второй смены концепта, где 1 (вернхняя кривая) без МОСК, 2 (нижняя кривая) – с МОСК

Две вертикальных линии на рисунке 4а (вблизи значений 20000 и 40000 по оси абсцисс) обозначают моменты смены концепта. На начальном участке вероятность ошибки находится на уровне единицы, что обусловлено необходимостью получить на этапе обучения модели классификации не менее 2000 элементов. Использование модели обнаружения смены концепта позволяют снизить примерно на 5% вероятность ошибки классификации.

Как видно из графиков на рисунках 4б и 4в, модели классификации, использующие МОСК на основе коэффициентов затухания, начинают хуже работать сразу после смены концепта. Модели классификации, не использующие МОСК, допускают на 5–7% больше ошибок, что подтверждает возможность обнаружения изменений в потоковых данных только по значениям атрибутов без использования априорных знаний об истинных классах обрабатываемых объектов.

Выводы

Разработан алгоритм обнаружения смены концепта приложений, основанный на анализе изменений статистических характеристиках атрибутов или заметного снижения качества классификации анализируемых приложений. В качестве базовой модели обнаружения смены концепта анализируемых приложений использованы ИНС типа автокодировщик.

Предлагаемый метод использует несколько АК, каждый из которых обучен на соответствующем классе при отсутствии смены концепта.

Показано, что если АК обучен только на доброкачественных экземплярах, то он сможет реконструировать нормальные наблюдения, но не может реконструировать аномальные наблюдения (неизвестные понятия). В результате, когда АК фиксирует существенную ошибку восстановления, это классифицирует данные наблюдения как аномальные. Наличие смены концепта определяется с помощью оценок ошибок восстановления анализируемых приложений и превышения пороговых значений.

Входящий пакетный поток передается АК на выходе которого вычисляются потери при восстановлении. вычисляются ошибки восстановления и превышения и сравниваются с соответствующими порогами экземпляра, группы и подсчета. Если два пакет превышает два порога, генерируется предупреждение, а если три последовательных пакета превышают оба порога, то подтверждается смена концепта.

ГЛАВА 4 ИССЛЕДОВАНИЕ ПОТОКОВОЙ КЛАССИФИКАЦИИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ АНАЛИЗА СЕТЕВОГО ТРАФИКА

В главе представлены результаты *исследования потоковой классификации мобильных приложений на основе анализа сетевого трафика, полученные* с помощью разработанного программного комплекса CAT. Он включает сервер баз данных, сервер приложений, Web-приложение и клиентское программное обеспечение (ПО) для мобильных устройств под управлением ОС Android.

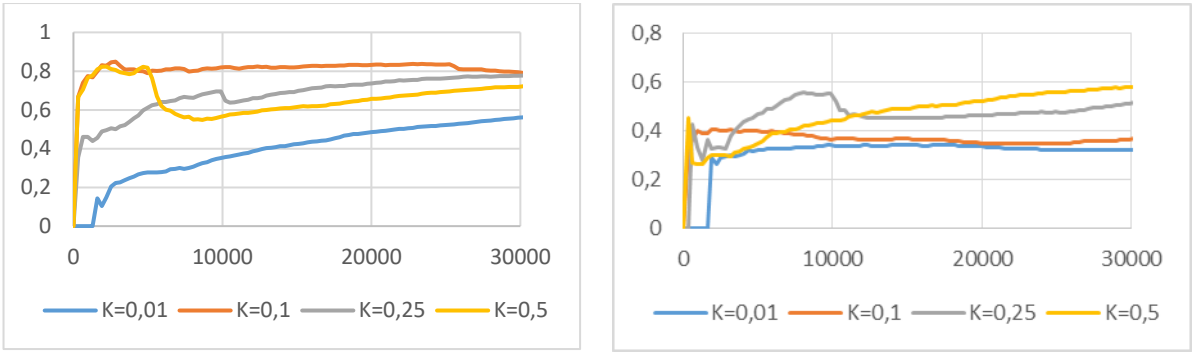
4.1 Классификация мобильных приложений на основе анализа сетевого трафика в потоковом режиме

Проведен сравнительный анализ известных алгоритмов онлайн-классификации мобильных приложений [113, 114, 115] на основе анализа сетевого трафика: Adaptive Random Forest, Hoeffding Adaptive Tree, K nearest neighbors, Oza Bagging в режимах с «конечной» и «бесконечной» памятью на примере сетевого трафика 6 мобильных приложений: IGM, SP, MI_RU, SB, HSN, PKB. Для каждого приложения исследовалось 5000 TCP сеансов, часть из которых была захвачена с момента начала «рукопожатия», другая – уже во время передачи информации.

Анализировались различные случаи распределения интенсивности сетевого трафика. В первом случае приложения поступали равномерно и непрерывно, формируя «равномерное» распределение сетевого трафика; во втором, более практичном случае, анализируемые приложения появлялись со случайной интенсивностью и длительностью.

Из экспериментальной последовательности потоковых данных формировались 100 периодов длительностью $T = T_{\text{об}} + T_{\text{тест}}$, где $T_{\text{об}}$ – длительность интервала обучения, $T_{\text{тест}}$ – длительность интервала тестирования модели классификации. При этом удобным оказалось введение в рассмотрение параметра $\text{K} = \frac{T_{\text{об}}}{T}$. Формирование показателей качества осуществлялось на основе обработки $n \ll 100$ периодов T в окне размера $W = n \cdot T$.

При исследовании квазистационарного режима (равномерного распределения входных данных во времени). Получено, что наилучшие результаты качества классификации демонстрирует алгоритм ARF. На рисунке 4.1 приведены результаты исследования показателя $Precision$, характеризующего качество модели классификации ARF в режиме равномерного поступления сетевого трафика 2 приложений и накопления данных.



а) б)

Рисунок 4.1 – Усредненные значения показателя Precision для случая равномерного поступления данных: а) ISG; б) SP.

Для случая неравномерного поступления данных получены оценки эффективности алгоритма ARF на всем потоке сетевого трафика. На текущем отрезке времени проводилась оценка характеристик в режиме «с конечной памятью» и запоминанием результатов классификации.

Исследования показали, что алгоритм ARF «хорошо» справляется с задачами классификации как при равномерном, так и при неравномерном поступлении сетевого трафика. Для сетевого трафика относительно равномерной интенсивности рекомендуется значение параметра $K \approx 0{,}5$, для неравномерного сетевого трафика наилучшие результаты обеспечивает $K \approx 0{,}8$. Кроме того, в последнем случае лучшее качество обеспечивает режим накопления с фиксированным размером окна порядка $W = 20T$.

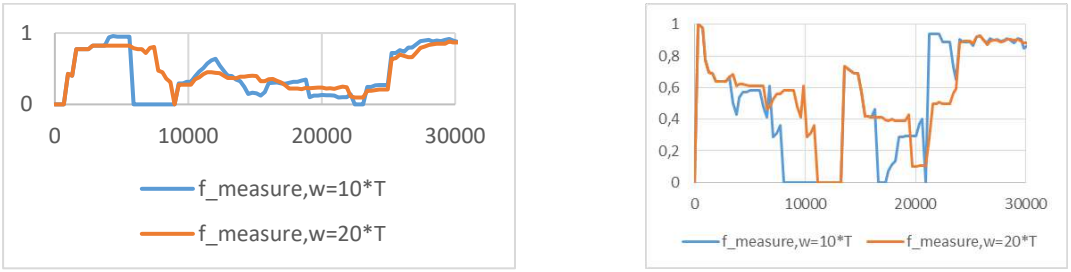
Указанный подход был реализован путем модификации известного алгоритма ARF. Для обнаружения смены концепта в MARF используется МОСК на основе автокодировщиков, не требующая истинных меток. Использование такой МОСК позволяет обнаруживать смену концепта не только на этапе обучения, но и на этапе предсказания.

Следует отметить, что при работе с квазистационарными потоками данных, в которых отсутствует смена концепта, часть алгоритма, работающая с «запасными деревьями», как и в алгоритме ARF, не используется.

Типично обучаемая модель классификации обновляется каждый раз при обнаружении смены концепта, что приводит к ухудшению его эффективности, поскольку после сброса требуется время на его обучение. Для устранения этого недостатка в MARF, так же, как и в ARF используются «запасные деревья». Они обучаются параллельно со всем ансамблем, однако не участвуют в предсказании до того момента, пока не обнаруживается смена концепта, и используемое дерево перестанет быть эффективным. В этот момент дерево, для которого зафиксирована смена концепта, заменяется ассоциированным с ним запасным.

Использование MARF позволило осуществлять классификацию в 2-3 раза быстрее, чем алгоритм RF, а также Hoeffding Adaptive Tree, K nearest neighbors, Oza Bagging. Это делает его предпочтительным для решения задач классификации мобильных приложений на основе анализа сетевого трафика в реальном масштабе времени.

На рисунке 4.2 показано, что эффективность классификации существенно зависит от свойств реального потока данных. Для параметра $K = 0{,}83$ качество модели классификации приближается к 99%.



а) б)

Рисунок 4.2 – Зависимости показателей F-мера для неравномерного поступления данных приложений: а) ISG; б) SP.

4.2 Разработка программного комплекса «Система анализа трафика»

4.2.1 Анализ предметной области

Для автоматизации процесса исследования алгоритмов классификации мобильных приложений на основе анализа сетевого трафика требуется разработка программного комплекса [116, 117], позволяющего в автоматическом режиме собирать с мобильных устройств пакеты сетевого трафика и сохранять их в базу данных; группировать пакеты сетевого трафика в потоки; по запросу пользователя формировать наборы данных с заданными характеристиками (количество потоков конкретного приложения: наличие фонового трафика; формирование набора данных на основе уже созданного набора с добавлением новых потоков, исключая повторения).

Программный комплекс должен:

- вычислять атрибуты потока с возможностью легкого добавления новых вычисляемых атрибутов;
- создавать и обучать модели классификации мобильных приложений путём анализа сетевого трафика с заданными набором данных, атрибутами и алгоритмом обучения;
- использовать кросс-валидацию;
- использовать online классификацию;
- применять алгоритмы выбора атрибутов;
- единообразно работать со сторонними библиотеками машинного обучения (такими как WEKA, MOA и т.п.) и собственными алгоритмами машинного обучения;
- получать численные результаты оценки эффективности моделей классификации;
- осуществлять управление процессом исследования с помощью мобильного клиента и Web-браузера;
- получать с помощью Web-браузера информацию о наборах данных, моделях классификации;

– в автоматическом режиме определять приложение-источник по пакетам сетевого трафика.

Взаимодействие приложений с базой данных показано на рисунке 4.3

Рисунок 4.3 – Взаимодействие приложений с базой данных

На основании сформулированных требований можно выделить следующие сущности создаваемого ПО:

– Пакет – хранит информацию о пакетах сетевого трафика
– Поток – хранит информацию о потоке сетевого трафика
– Приложение – хранит информацию о мобильном приложении
– Набор потоков – хранит информацию о наборе потоков, которые в дальнейшем будут использованы для обучения и проверки модели классификации
– Атрибут – хранит информацию об атрибутах классификации, которые могут быть вычислены
– Кеш атрибутов – хранит информации о значении атрибутов классификации для потоков. Используется для сокращения времени вычисления атрибутов
– Набор атрибутов – хранит информацию об атрибутах, используемых при классификации для описания потока сетевого трафика
– Сеанс фильтрации атрибутов – хранит информацию о сеансах фильтрации атрибутов классификации

- Параметры – хранит информацию о значениях параметров алгоритма выбора атрибутов, алгоритма оценки, используемого при фильтрации атрибутов, алгоритма классификации
- Обработчик сетевого трафика – общая сущность для моделей классификации
- Матрица ошибок – хранит матрицы ошибок всех проведённых экспериментов
- Ячейка матрицы ошибок – составная часть матрицы ошибок
- Эксперимент – хранит информацию о проведённых экспериментах (классификация)
- Пользователь – хранит информацию о пользователях системы
- Сеансы пользователя – хранит информацию обо всех активных сеансах пользователей

Сеанс фильтрации атрибутов может иметь несколько параметров алгоритма фильтрации атрибутов и несколько параметров алгоритма оценки, указывает на один или несколько начальных атрибутов и на результирующий набор атрибутов, связан с набором потоков и имеет владельца.

Каждый обработчик сетевого трафика (модель классификации) может иметь несколько параметров, относящиеся к алгоритму классификации, связан с набором атрибутов и имеет владельца.

Поток может включать в себя один или несколько пакетов, связан с приложением и имеет владельца.

Набор потоков включает в себя один или несколько потоков сетевого трафика, а также может включать несколько потоков фонового трафика и имеет владельца.

Каждый пакет сопоставлен с приложением.

Набор атрибутов включает в себя один или несколько атрибутов и имеет владельца.

Кэш атрибутов связан с потоком и атрибутом.

Каждый эксперимент включает в себя один или несколько обработчиков сетевого трафика (моделей классификации), набор потоков и имеет владельца.

Матрица ошибок состоит из одной или нескольких ячеек и сопоставлена с набором данных и обработчиком сетевого трафика (моделью классификации). Каждая ячейка матрицы ошибок ссылается на ожидаемое и полученное в результате классификации приложение.

Каждый пользователь может иметь одну или несколько сессий.

4.2.2 Инфологическое проектирование

На основании анализа предметной области построена инфологическая модель предметной области, включающая 15 сущностей и 29 связей. ER-диаграмма, описывающая сущности, атрибуты сущностей и связи сущностей представлена на рисунке 4.4**Ошибка! Источник ссылки не найден.**.

Сущность ConfusionMatrix (Матрица ошибок) хранит информацию наборе потоков и обработчике сетевого трафика, для которых данная матрица ошибок была получена, а также о наборе ячеек.

Сущность ConfusionMatrixCell (Ячейка матрицы ошибок) представляет ячейку матрицы ошибок и хранит информацию о количестве потоков, которые были классифицированы как принадлежащие приложению, указанному в поле predictedApplication, но на самом деле принадлежат приложению, указанном в поле requiredApplication.

Сущность Application (Приложение) хранит информацию о названии приложения, которое взято из магазина приложений Google Play и имени пакета – уникальном строковым идентификатором приложения.

Сущность Flow (Поток) хранит информацию о дате создания потока, IP-адресе и порте источника и назначения, версии операционной системы клиента, типе соединения (мобильная сеть или сеть Wi-Fi), временных метках первого и последнего пакетов в потоке, количестве и наборе пакетов, информацию о приложении, которому принадлежит данный поток, владельце потока, а также признак, указывающий активно соединение или уже закрыто.

Сущность Packet (Пакет) хранит бинарное представление IP-датаграмы, приложение, которому принадлежит пакет, а также временную метку, по которой можно определить дату и время генерации датаграмы.

Сущность FlowSet (Набор потоков) хранит название набора данных, признак, с помощью которого можно определить, является ли набор скрытым или нет, набор потоков, набор потоков, играющих роль фонового трафика, а также о владельце набора потоков.

Сущность Processor (Обработчик сетевого трафика) хранит название обработчика сетевого трафика, тип (модель классификации), алгоритм классификации, данные, полученные в результате обучения, набор атрибутов, по которым осуществляется обучение и тестирование, параметры алгоритма классификации, данные о владельце обработчика сетевого трафика.

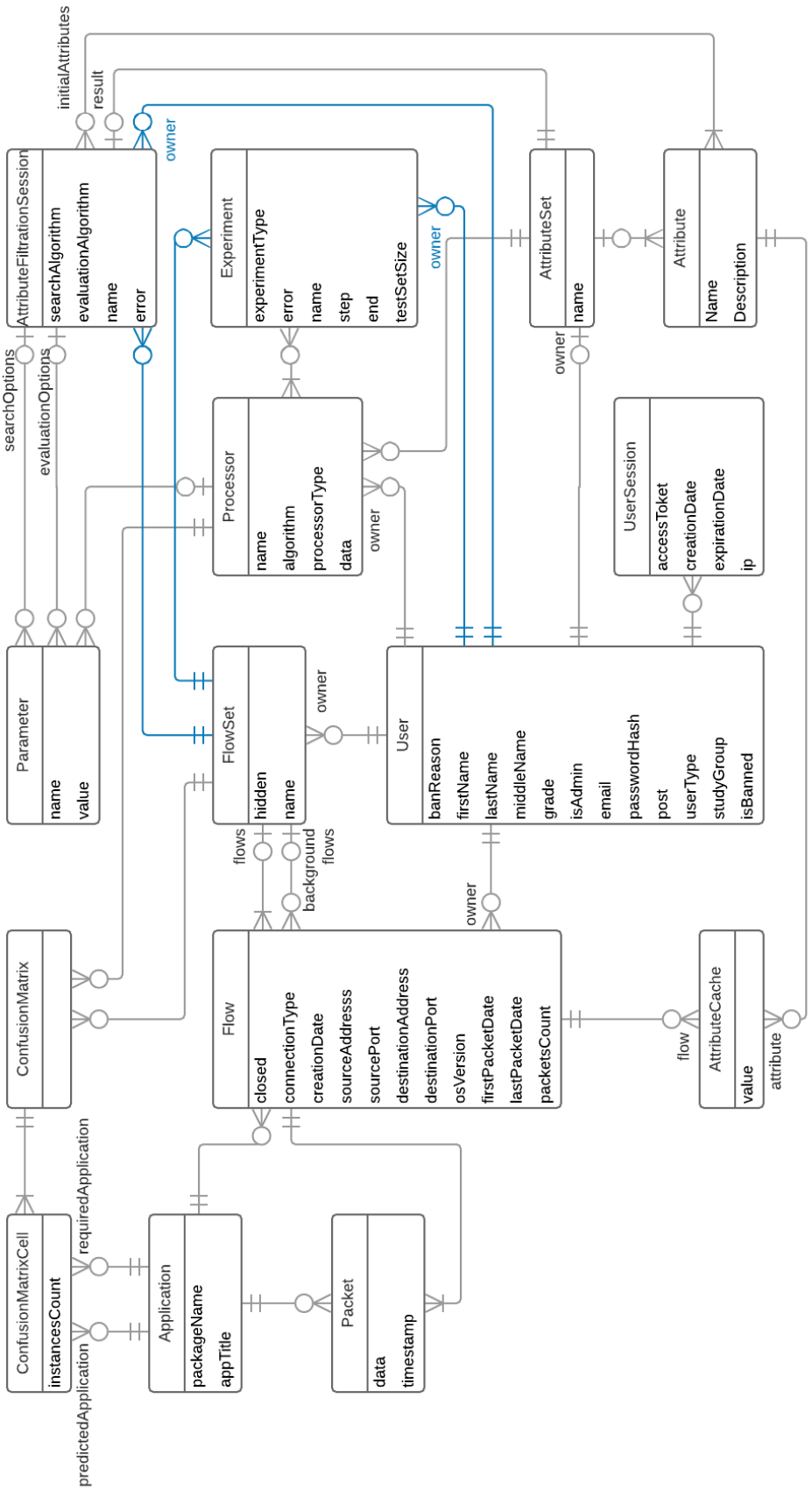


Рисунок 4.4 – Инфологическая модель предметной области «Классификация мобильных приложениий»

Сущность AttributeFiltrationSession (Сессия фильтрации атрибутов) хранит информацию о названии сессии, об алгоритме отбора атрибутов и алгоритме оценки, о параметрах алгоритмов отбора атрибутов и оценки, наборе потоков, на котором производится фильтрация атрибутов, список начальных атрибутов, результирующий набор атрибутов, информация о владельце набора атрибутов, а также информация об ошибках.

Сущность Parameter (Параметры) хранит информацию о паре ключ-значение, используется для хранения параметров алгоритмов отбора атрибутов, оценки и классификации.

Сущность AtributeSet (Набор атрибутов) хранит информацию о названии набора атрибутов, входящих в состав набора атрибутах и владельце набора атрибутов.

Сущность Attribute (Атрибут) хранит название и описание атрибута.

Сущность AttributeCache (Кэш атрибутов) хранит вычисленное значение конкретного атрибута для конкретного потока.

Сущность Experiment (Эксперимент) хранит информацию о наборе потоков, с которым проводится эксперимент, обработчике сетевого трафика, владельце эксперимента, названии и типе эксперимента, наличии ошибок, размере тестовой выборки, общие числовые характеристики, зависящие от типа эксперимента, такие как шаг и конечное значение изменяемого параметра.

Сущность User (Пользователь) хранит информацию о пользователе системы: фамилию, имя, отчество, ученую степень, роль пользователя (студент или преподаватель), группу для студентов, занимаемую должность для преподавателей, адрес электронный почты и хеш пароля для осуществления процесса аутентификации, признак того, что пользователь является администратором, признак того, что пользователь заблокирован и причина блокировки.

Сущность UserSession (Сеансы пользователя) хранит информацию о текущем сеансе пользователя: информация о пользователе, дата начала и окончания сеанса, IP-адрес, с которого был установлен сеанс связи и токен доступа.

4.2.3 Проектирование базы данных: даталогическое и физическое проектирование

В ходе даталогического проектирования базы данных для реализации связей многие ко многим и связей с необязательным классом принадлежности с обеих сторон были созданы вспомогательные сущности. Общее число сущностей увеличилось до 21. Перечень сущностей представлен в таблице 4.1.

Таблица 4.1 – Сущности базы данных

№	Сущность	Назначение
1	apps	Хранит информацию о приложениях
2	confusionmatrices	Хранит информацию о матрицах ошибок
3	confusionmatrixcells	Хранит информацию о ячейках матрицы ошибок
4	evaluationoptions	Хранит информацию о значениях параметров алгоритма оценки, используемого при фильтрации атрибутов
5	experimentresultpoints	Хранит информацию о точках экспериментов
6	experiments	Хранит информацию об экспериментах
7	flows	Хранит информацию о потоках
8	flowsinset	Хранит информацию о связи потоков и наборов потоков
9	flowssets	Хранит информацию о наборах потоков
10	optionvalue	Хранит информацию о значениях параметров
11	packets	Хранит информацию о пакетах
12	processoroptions	Хранит информацию о связи моделей классификации и значений параметров
13	processors	Хранит информацию о моделях классификации
14	processorsinexperiments	Хранит информацию о связях экспериментов и моделей классификации
15	searchoptions	Хранит информацию о связях сеансов фильтрации атрибутов и значений параметров алгоритма отбора атрибутов
16	user	Хранит информацию о пользователях
17	usersession	Хранит информацию о сессиях пользователей
18	attributesfiltrationsessions	Хранит информацию о сеансах фильтрации атрибутов
19	attributevalues	Хранит информацию о значения атрибутов для потоков
20	backgroundflows	Хранит информацию о связях фоновых потоках и наборах потоков
21	attributesset	Хранит информацию о наборах атрибутов

За хранение пакетов сетевого трафика мобильных приложений и наборов данных отвечают сущности apps, flows, flowsinset, flowsets, packets, backgroundflows.

Для развёртывания программного комплекса был выбран сервер IMB под управлением операционной системы Microsoft Windows Server 2016 Standard. В качестве сервера базы данных была выбрана бесплатная СУБД MySQL 5.7.

На рисунке 4.5 показана схема базы данных.

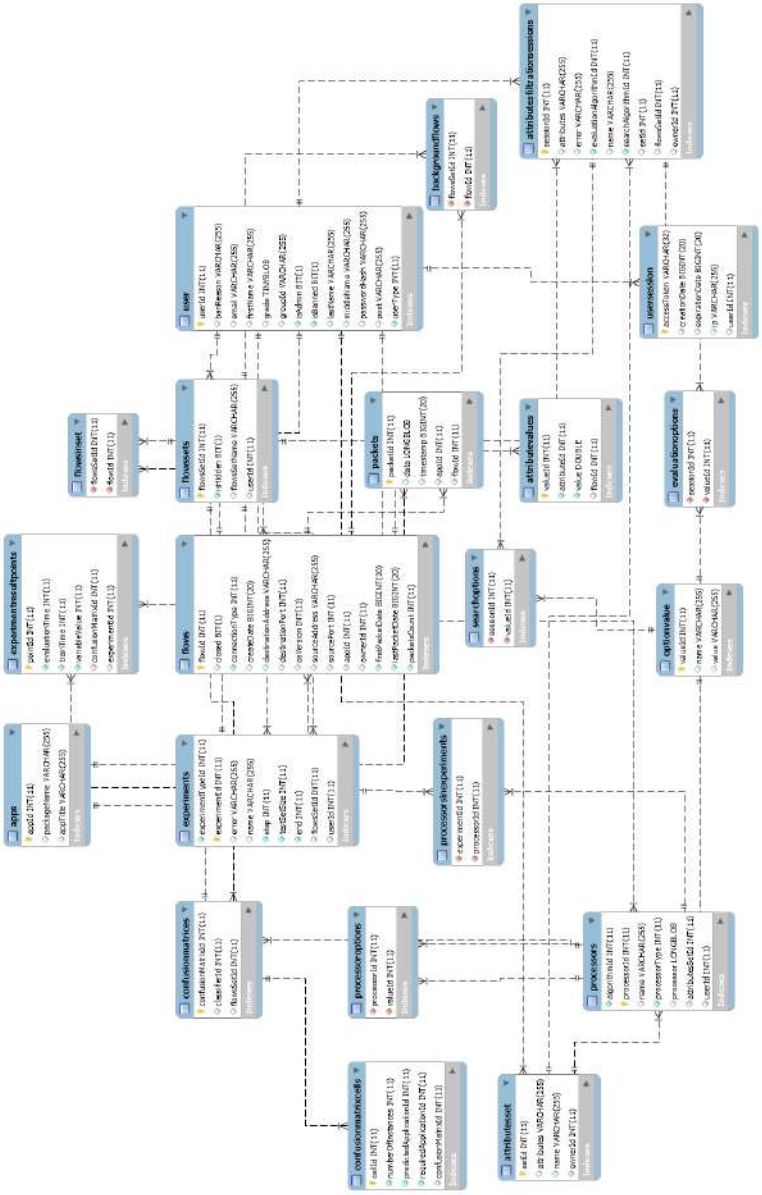


Рисунок 4.5 – Схема базы данных

4.2.4 Проектирование и разработка серверного программного обеспечения

Для взаимодействия с разработанной базой данных с использованием технологий Java Enterprise Edition было разработано корпоративное приложение,

обеспечивающие возможность накопление пакетов и потоков сетевого трафика выбранных мобильных приложений, управления наборами потоков и проведения экспериментов. В ходе разработки приложения, по инфологической модели предметной области были созданы классы-сущности, и Web-сервис, с помощью которого с приложением могут удалённо взаимодействовать клиенты. Перечень поддерживаемых HTTP-запросов представлен в таблице 4.2.

Весь функционал, описанный в таблице 4.2 реализован с использованием EJB-компонентов, организующих получение и сохранение сущностей из базы данных. Обучение и тестирование моделей классификации осуществляется с использованием библиотек WEKA и MOA. Разработанные абстракции моделей классификации позволяют использовать любые другие библиотеки, в том числе написанных на других языках программирования, либо реализовывать собственные алгоритмы классификации.

Таблица 4.2 – Перечень поддерживаемых HTTP-запросов к Web-службе

Метод	Путь	Назначение
Администрирование		
PUT	/admin/ban	Блокировка доступа к API заданному пользователю
GET	/admin/users	Получение информации о всех зарегистрированных пользователях
Управление пользователями		
POST	/register	Регистрация пользователя
POST	/auth	Авторизация пользователя
GET	/users/current	Получение информации о текущем пользователе
Работа с атрибутами		
GET	/attributes	Получение всех атрибутов
GET	/attributes/filtration/sessions	Получение списка сеансов фильтрации атрибутов текущего пользователя
DELETE	/attributes/filtration/sessions/{id}	Удаление сессии фильтрации атрибутов с идентификаторов id
POST	/attributes/sets	Создание набора атрибутов
GET	/attributes/sets	Получение наборов атрибутов текущего пользователя
DELETE	/attributes/sets/{id}	Удаление набора атрибутов с идентификаторов id
GET	/attributes/sets/all	Получение всех наборов атрибутов, принадлежащих указанному пользователю

Метод	Путь	Назначение
POST	/attributes/sets/filter	Фильтрация атрибутов
GET	/evaluation-algorithms	Получение алгоритмов оценки атрибутов
GET	/search-algorithms	Получение всех доступных для использования алгоритмов выбора атрибутов
Работа с пакетами и потоками		
GET	/flows/count	Получение информации о количестве соединений по всем приложениям
GET	/flows/count/{appId}	Получение количества потоков по приложению с идентификаторов appId
GET	/flows/count/filter	Получение количества потоков по приложениям с учетом минимального количества пакетов в потоке
PUT	/packet	Точка приема перехваченного сетевого трафика
GET	/packets/count	Получение информации о количестве IP-дейтаграмм по всем приложениям
Работа с наборами данных		
GET	/applications	Получение списка собранных приложений
PUT	/dataset	Создание набора данных
DELETE	/dataset/{id}	Удаление уже созданных наборов данных с идентификаторов id
GET	/datasets	Получение наборов данных текущего пользователя
GET	/datasets/all	Получение всех созданных наборов данных всех пользователей
Работа с моделями классификации		
POST	/classifiers	Создание модели классификации
GET	/classifiers/algorithms	Получение алгоритмов классификации
Работа с моделями классификации		
GET	/processors	Получение списка моделей классификации
DELETE	/processors/{id}	Удаление модели классификации с идентификаторов id
Работа с экспериментами		
GET	/experiments	Получение списка экспериментов
DELETE	/experiments/{id}	Удаление уже созданного эксперимента с идентификаторов id
GET	/experiments/{id}	Получение информации об эксперименте с идентификаторов id
GET	/experiments/all	Получение созданных экспериментов пользователя
POST	/experiments/cross-validation	Создание эксперимента кросс-валидации
POST	/experiments/one-set-many-processors	Создание эксперимента "Один набор - много алгоритмов"
POST	/experiments/one-set-one-processor	Создание эксперимента "Один набор - один алгоритм"
POST	/experiments/packets-incrementation	Создание эксперимента "Увеличение пакетов"
GET	/experiments/status	Получение состояний выполнения всех экспериментов

Метод	Путь	Назначение
GET	/experiments/status/{id}	Получение статуса эксперимента с идентификаторов id
DELETE	/experiments/stop/{id}	Остановка эксперимента с идентификаторов id

Сбор сетевого трафика на мобильном устройстве осуществляется с помощью разработанного клиентского программного обеспечения. Перехват пакетов сетевого трафика осуществляется с помощью прикладного программного интерфейса для построения виртуальных частных сетей. Определение приложения, которому принадлежит сетевой трафик, производится в два этапа. На первом этапе с помощью псевдофайлов /proc/net/tcp и /proc/net/udp, предоставляемые операционной системой Linux, определяется идентификатор пользователя Linux, которому принадлежит сетевое подключение. На втором этапе с помощью класса PackageManager, входящего в Android API, по идентификатору пользователя Linux определяется приложение (имя пакета и название). Собранная информация отправляется на сервер с использованием описанного выше REST API.

4.2.5 Проектирование и разработка мобильного приложения для сбора сетевого трафика

Для проведения эксперимента и формирования исходных данных было установлено на мобильное устройство, под управление операционной системой Android [19] версии 4.4, специализированное программное обеспечение “Анализатор трафика”. На рисунке 4.6 представлен процесс подключения к серверу с использованием данного программного обеспечения. На рисунках 4.7–4.8 представлены процессы регистрации и авторизации на сервере.

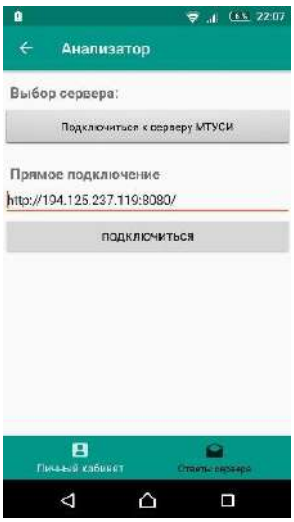


Рисунок 4.6 – Экран подключения к серверу

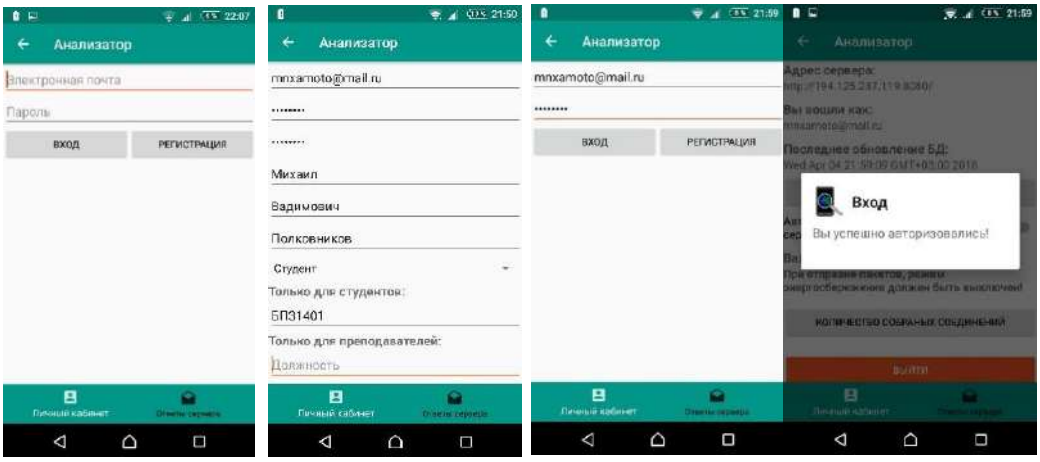


Рисунок 4.7 – Регистрация и авторизация пользователя

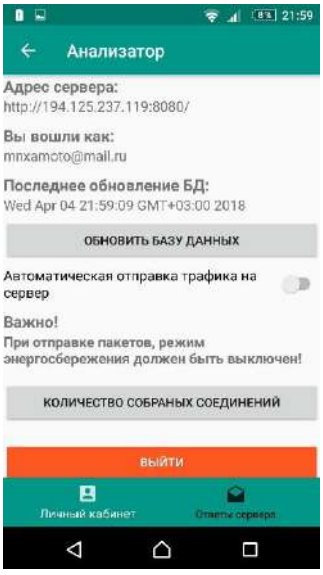


Рисунок 4.8 – Экран «Личный кабинет»

На рисунках 4.9–4.10 представлен процесс настройки данного программного обеспечения для перехвата сетевого трафика по указанным приложениям и отправки его на сервер.

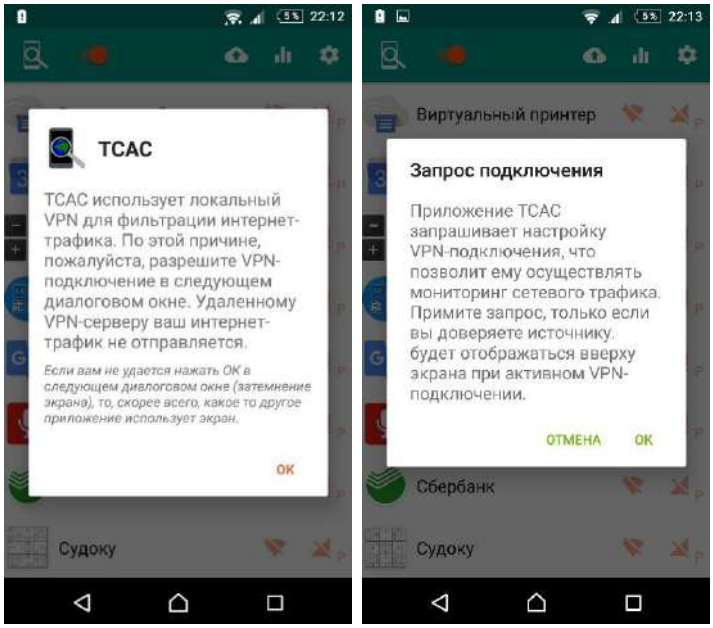


Рисунок 4.9 – Разрешение приложению перехватывать сетевой трафик других приложений

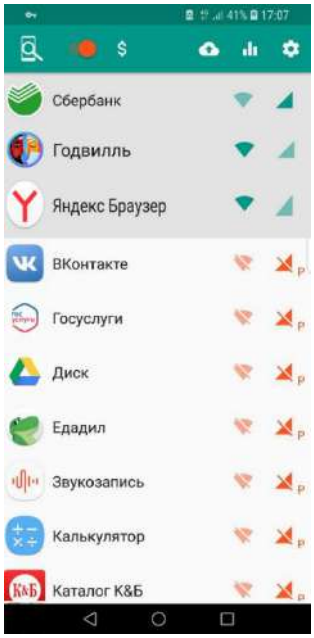


Рисунок 4.10 – Выбор приложений для сбора сетевого трафика

На рисунке 4.11 представлен процесс перехвата сетевого трафика и отправки его на сервер.

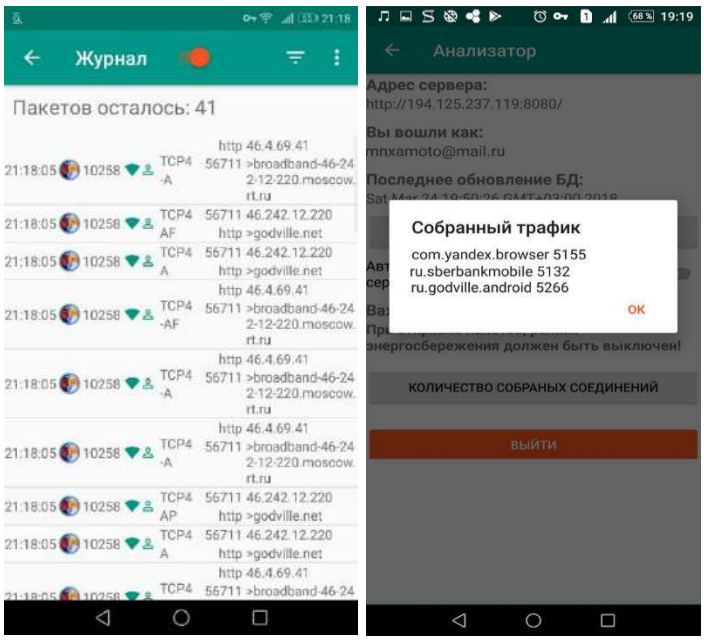


Рисунок 4.11 – Процесс сбора сетевого трафика

На разработанный программный комплекс получено свидетельство о регистрации программы для ЭВМ [118]. Копия свидетельства представлена в приложении В. Результаты диссертационного исследования, полученные с использованием разработанного программного обеспечения используются учебном процессе ФГБОУ ВО МТУСИ (приложение Г) и АО «Лаборатория Касперского» (приложения Д).

Выводы

Adaptive Random Forest работает на рассмотренном наборе данных лучше, чем остальные алгоритмы. На втором месте, по качеству, после него находится алгоритм K Nearest Neighbors.

Лучшее время обучения имеет алгоритм K Nearest Neighbors, что обусловлено тем, что в фазе обучения, алгоритм просто сохраняет наблюдения во внутреннем хранилище. При малом времени обучения алгоритм имеет длительную фазу тестирования. Причина данного явления заключается в том, что для классификации одного объекта модели классификации требуется перебрать все наблюдения, сохраненные алгоритмом обучения в обучающей фазе. для

достижения максимального качества классификации, алгоритму достаточно обучающей выборки размером 20000 потоков.

Время обучения для алгоритма Adaptive Random Forest растет линейно при увеличении размера обучающей выборки. Время тестирования держится, примерно, на одном уровне, за исключением всплеска ближе к концу графика

Наибольшее время обучения имеет алгоритм Adaptive Random Forest. При этом время тестирования практически сходится с временем тестирования алгоритма Hoeffding Adaptive Tree, основанном на алгоритме Hoeffding Tree.

Метрики качества для алгоритма Adaptive Random Forest возрастают нелинейно с увеличением размера обучающей выборки. в большинстве случаев, достаточно 40 пакетов для стабильной классификации приложений.

Наиболее точные результаты при равномерном поступлении сетевого трафика получены при $K = 0{,}5$. Непрерывное накопление результатов приводит к более стабильным показателям точности. Наилучшую точность по-прежнему обеспечивает соотношение $K = 0{,}5$.

Значения параметра Точность сильно варьируется в зависимости от момента поступления сетевого трафика. Кроме того, появляются нулевые значения точности, из-за отсутствия потоков приложения в определённый момент. Такие показатели качества неприемлемы для классификации. При непрерывном накоплении результатов наилучшие результаты точности достигаются при параметре $K = 0{,}83$. Для анализа качества классификации и сокращения накладных расходов рекомендуется применять метод скользящего окна размером $W = 20T$

В ходе главы было проведено инфологическое, даталогическое и физическое проектирование базы данных, в ходе которых было выделено 15 сущностей и 29 связей. На основании выделенных сущностей было создано 21 таблица. Разработана Web-служба и мобильный клиент

ЗАКЛЮЧЕНИЕ

В диссертационном исследовании решена актуальная научная задача разработки программно-методического инструментария для обнаружения и классификации в потоковом режиме мобильных приложений путём анализа сетевого трафика. Получены следующие результаты.

1) Предложена методика отбора значимых атрибутов, обеспечивающая высокую достоверность и снижение вероятности ложной классификации до 2,5 раз. Это позволило применить классификацию мобильных приложений в потоковом режиме, которая является инвариантной по отношению к разным типам сетевого трафика.
2) Для обеспечения достаточно высокой достоверности классификации мобильных приложений, осуществляющих шифрование сетевого трафика, размер обучающей выборки достаточно ограничить 300 потоками с 16-58 пакетами в каждом и числом информативных атрибутов не более 13 (в зависимости от приложения), в то время как общепринятые подходы предполагают использование данных всего потока.
3) Разработан новый алгоритм классификации мобильных приложений, осуществляющих распространение противоправного, вредоносного и нежелательного контента, в условиях априорной неопределенности и неконтролируемого фонового трафика посредством последовательного включения АК и типовой модели классификации, что позволило повысить достоверность классификации приложений на 5-7% по сравнению с типовыми алгоритмами, не требуя разметки фоновых приложений.
4) Разработана статистическая модель обнаружения смены концепта при классификации мобильных приложений на основе анализа сетевого трафика, отличающаяся от известных тем, что АК включён в качестве базовой модели обнаружения, а момент наступления смены концепта определяется по превышению порогов ошибок восстановления атрибутов

мобильных приложений, что позволяет повысить точность обнаружения смены концепта в реальном времени до 10%.

5) Разработан алгоритм MARF (модификация алгоритма ARF) со встроенной моделью обнаружения смены концепта, позволяющей обнаруживать смену концепта не только во время обучения, но и во время предсказания, т.к. не использует истинные метки и осуществляющий классификацию в 2–3 раза быстрее, чем известные алгоритмы (RF, HAT, KNN, OB) при выборе показателя скважности K≈0,5; для неравномерного сетевого трафика лучшие результаты обеспечивает K≈0.8, при котором достоверность классификации приближается к 99%.
6) Разработан и реализован ПК, включающий сервер баз данных, сервер приложений, Web-приложение и ПО для мобильных устройств под управлением ОС Android. Он позволяет: собирать с мобильных устройств пакеты сетевого трафика и сохранять их в БД; группировать пакеты сетевого трафика в потоки; по запросу пользователя формировать наборы данных с заданными характеристиками, автоматизировать процесс классификации мобильных приложений путём анализа сетевого трафика и др.

Полученные результаты позволяют заключить, что все поставленные задачи решены и цель диссертационного исследования достигнута.

СПИСОК ЛИТЕРАТУРЫ

1. Федеральный закон от 27.07.2006 N 149-ФЗ (ред. от 29.12.2022) "Об информации, информационных технологиях и о защите информации" (с изменениями и дополнениями, вступившими в силу с 01.03.2023).
2. Федеральный закон от 29 декабря 2006 года N 244-ФЗ «О государственном регулировании деятельности по организации и проведению азартных игр и о внесении изменений в некоторые законодательные акты Российской Федерации».
3. Федеральный закон от 11 ноября 2003 года N 138-ФЗ «О лотереях».
4. Федеральный закон от 20 апреля 1995 года N 45-ФЗ «О государственной защите судей, должностных лиц правоохранительных и контролирующих органов».
5. Федеральный закон от 20 августа 2004 года N 119-ФЗ «О государственной защите потерпевших, свидетелей и иных участников уголовного судопроизводства».
6. 33 лучших хакерских приложений для Android устройств [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://dzen.ru/a/X7jATlcrhiV1Dp96, свободный.
7. Приложения в Google Play – Orbot Прокси в комплекте с Tor [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://play.google.com/store/apps/details?id=org.torproject.android&pli=1, свободный.
8. Приложения в Google Play – Fing - Сетевые инструменты [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://play.google.com/store/apps/details?id=com.overlook.android.fing, свободный
9. Приложения в Google Play – DriveDroid [Электронный ресурс]. – Электрон. дан. – Режим доступа:

https://play.google.com/store/apps/details?id=com.softwarebakery.drivedroid, свободный.

10. Приложения в Google Play – Pixelknot [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://play.google.com/store/apps/details?id=info.guardianproject.pixelknot, свободный.
11. Приложения в Google Play – WIFI WPS WPA TESTER [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://play.google.com/store/apps/details?id=com.tester.wpswpatester, свободный.
12. Приложения в Google Play – Change My MAC - изменить MAC-a [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://play.google.com/store/apps/details?id=net.xnano.android.changemymac, свободный.
13. [APP][2.2+][ROOT] Market Helper - spoof your device to download incompatible apps | XDA Forums [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://xdaforums.com/t/app-2-2-root-market-helper-spoof-your-device-to-download-incompatible-apps.2146216/, свободный.
14. GitHub - DesignativeDave/androrat: Remote Administration Tool for Android devices [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://github.com/DesignativeDave/androrat, свободный.
15. Penetration Testing for Mobile Applications Pentesting Toolkit | zANTI [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://www.zimperium.com/zanti-mobile-penetration-testingPeP/, свободный.
16. Google Code Archive - Long-term storage for Google Code Project Hosting [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://code.google.com/archive/p/anmap/downloads, свободный.
17. Droid Pentest - Browse Files at SourceForge.net [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://sourceforge.net/projects/droidpentest/files/, свободный.

18. [APP][BETA]USB Cleaver - USB Password Recovery Tool | XDA Forums [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://xdaforums.com/t/app-beta-usb-cleaver-usb-password-recovery-tool.1656497/, свободный.
19. AppUse - AppSec Labs [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://appsec-labs.com/appuse/#av_section_5, свободный.
20. Get Kali | Kali Linux [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://www.kali.org/get-kali/#kali-mobile, свободный.
21. GitHub - intercepter-ng/intercepter-ng.github.io: mirror [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://github.com/intercepter-ng/intercepter-ng.github.io, свободный.
22. Шелухин, О. И., Ерохин, С. Д., Ванюшина, А. В. Классификация IP-трафика методами машинного обучения / Под редакцией О. И. Шелухина. –М.:Горячая линия — Телеком, 2018. – 284 с.
23. Шелухин, О. И., Ерохин, С. Д., Полковников, М. В. Технологии машинного обучения в сетевой безопасности. – М.: Научно-техническое издательство «Горячая линия – Телеком», 2023. – 360 с. – ISBN 978-5-9912-0913-7.
24. Quinlan, J. R. Induction of decision trees // Machine learning. – 1986. – Vol. 1. – pp. 81–106.
25. Quinlan, J. R. C4.5: Programs for Machine Learning. – San Mateo, Calif.:Morgan Kaufmann Publishers, 1993. – pp. 302.
26. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A. Classification and Regression Trees. – London: Taylor & Francis, 1984. – pp. 368.
27. Kass, G. V. An Exploratory Technique for Investigating Large Quantities of Categorical Data // Journal of the Royal Statistical Society Series C. – 1980. – Vol. 29(2). – pp. 119–127.
28. Loh, W.-Y., Shih, Y.-S. Split selection methods for classification trees // Stat. Sinica. – 1997. – Vol. 7. – pp. 815–840.
29. Breiman, L. Bagging predictors // Machine learning. – 1996. – Vol. 24. – pp. 123–140.

30. Yoav, F., Schapire, R., Abe, N. A Short Introduction to Boosting // Journal Japanese Society For Artificial Intelligence. – 1999. – Vol. 14(5). – pp. 771-780.
31. Friedman, J. H. Stochastic gradient boosting // Computational Statistics and Data Analysis. – 2002. –Vol. 38(4). – pp. 367-378.
32. CatBoost — open-source gradient boosting library [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://catboost.ai/, свободный (дата обращения: 27.07.2023).
33. Welcome to LightGBM's documentation! — LightGBM 2.3.2 documentation [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://lightgbm.readthedocs.io/en/latest/, свободный (дата обращения: 27.07.2023).
34. XGBoost [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://xgboost.ai/, свободный (дата обращения: 27.07.2023).
35. Breiman, L. Random Forests // Machine Learning. – 2001. – Vol. 45(1). – pp. 5–32.
36. Liu, F. T., Ting, K. M., Zhou, Z-H. Isolation forest // Proceedings of the Eighth IEEE International Conference on Data Mining. – Los Alamitos CA USA, IEEE, 2008. – pp. 413–422.
37. Jayalakshmi, T., Santhakumaran, A. Statistical normalization and back propagationfor classification // International Journal of Computer Theory and Engineering. – IJCTE, 2011. – Vol. 3(1). – pp. 89-93.
38. Miao, Xie, Jiankun, Hu, Song, Han, Hsiao-Hwa, Chen. Scalable hypergrid k-nn-based online anomaly detection in wireless sensor networks // IEEE Transactions on Parallel and Distributed Systems. – 2013. – Vol. 24(8). – pp. 1661–1670.
39. Xiangzeng, Zhou, Lei, Xie, Peng, Zhang, Yanning, Zhang. An ensemble of deep neural networks for object tracking // IEEE International Conference on Image Processing. – 2014. –pp. 843–847.
40. Mahmood, Yousefi-Azar, Vijay, Varadharajan, Len, Hamey, Uday, Tupakula. Autoencoder-based feature learning for cyber securityapplications // 2017 International Joint Conference on Neural Networks. – 2017. – pp. 3854–3861.

41. Ahmad, Javaid, Quamar, Niyaz, Weiqing, Sun, Mansoor, Alam. A deep learning approach for network intrusion detection system // ICST Transactions on Security and Safety. – 2016. – Vol. 9(3). – pp. 21–26.
42. Mayu, Sakurada, Takehisa, Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. // MLSDA'14: Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis. – 2014. – pp. 4-11.
43. Gomes, H. M. et al. Adaptive random forests for evolving data stream classification // Machine Learning. – 2017. – V. 106(6). – pp. 1469– 1495.
44. Oza, Russell S. Online Bagging and Boosting // Proceedings of Artificial Intelligence and Statistics. – 2001. – pp. 105–112.
45. Wares, S., Isaacs, J., Elyan, E. Data stream mining: methods and challenges for handling concept drift // SN Applied Sciences. – 2019. – Vol. 1(11).
46. Gemaque, R. N., Costa, A. F. J., Giusti, R., Santos, E. M. dos. An overview of unsupervised drift detection methods // Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. – 2020. – Vol. 10(6).
47. Ditzler, G., Roveri, M., Alippi, C., Polikar, R. Learning in Nonstationary Environments: A Survey // IEEE Computational Intelligence Magazine. – 2015. – Vol. 10(4). – pp. 12–25.
48. Wald, A. Sequential Analysis. – DOVER PUBLICATIONS, INC, 1973. – pp. 212.
49. Page, E. S. Continuous Inspection Schemes // Biometrika. – 1954. – Vol. 41(1/2). – p. 100-115.
50. Bifet, A. Classifier concept drift detection and the illusion of progress // Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). – 2017. – Vol. 10246. – pp. 715–725.
51. Schlimmer, J. C., Granger, R. H. Incremental Learning from Noisy Data // Machine Learning. – 1986. – Vol. 1(3). – pp. 317–354.
52. Gama, J., Medas, P., Castillo, G., Rodrigues, P. Learning with drift detection // Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial

Intelligence and Lecture Notes in Bioinformatics). – 2004. – Vol. 3171. – pp. 286–295.

53. Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., Morales-Bueno, R. Early Drift Detection Method // 4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams. – 2006. – Vol. 6. – pp. 77–86.
54. Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P., Yao, X. Concept Drift Detection for Online Class Imbalance Learning. // The 2013 International Joint Conference on Neural Networks (IJCNN). – 2013.
55. Nishida, K., & Yamauchi, K. Detecting concept drift using statistical testing. // Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). – 2007. – Vol. 4755. – pp. 264–269.
56. Bifet, A., Gavaldà, R. Learning from time-changing data with adaptive windowing // Proceedings of the 7th SIAM International Conference on Data Mining. – 2007. – pp. 443–448.
57. Frías-Blanco, I., Campo-Ávila, J., Ramos, G., et al. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds // IEEE Transactions on Knowledge and Data Engineering. – 2015. – Vol. 27(3). – pp. 810-823.
58. Wang, H., Abraham, Z. Concept drift detection for streaming data // Proceedings of the International Joint Conference on Neural Networks. – 2015. – pp. 1-9.
59. Yu, S., Abraham, Z. Concept drift detection with hierarchical hypothesis testing // Proceedings of the 17th SIAM International Conference on Data Mining, SDM. – 2017. – pp. 768–776.
60. Barros, R. S. M., Cabral, D. R. L., Gonçalves, P. M., Santos, S. G. T. C. RDDM: Reactive drift detection method // Expert Systems with Applications. – 2017. – Vol. 90. – pp. 344–355.
61. Cabral, D. R. de L., Barros, R. S. M. de. Concept drift detection based on Fisher's Exact test // Information Sciences. – 2018. – Vol. 442–443. – pp. 220–234.

62. Pesaranghader, A., Viktor, H. L., Paquet, E. McDiarmid Drift Detection Methods for Evolving Data Streams // Proceedings of the International Joint Conference on Neural Networks. – 2018. – pp. 1-9.
63. Street, W. Nick, Kim, Y. S. A streaming ensemble algorithm (SEA) for large-scale classification // Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2001. – pp. 377–382.
64. Wang, H., Fan, W., Yu, P. S., Han, J. Mining concept-drifting data streams using ensemble classifiers // KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. – 2003. – pp. 226-235.
65. Brzeziński, D., Stefanowski, J. Accuracy updated ensemble for data streams with concept drift // International Conference on Hybrid Intelligent Systems. – 2011. – Vol. 6679. – pp. 155–163.
66. Kolter, J. Zico, Maloof, M. A. Dynamic weighted majority: An ensemble method for drifting concepts // Journal of Machine Learning Research. – 2007. – Vol. 8. – pp. 2755–2790.
67. Polikar, R., Udpa, L., Udpa, S. S., Honavar, V. Learn++: An incremental learning algorithm for supervised neural networks // IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews. – 2001. – Vol. 31(4). – pp. 497– 508.
68. Muhlbaier, M., Topalis, A., Polikar, R. Learn++.MT: A new approach to incremental learning // Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . – 2004. – Vol. 3077. – pp. 52–61.
69. Muhlbaier, M. D., Topalis, A., Polikar, R. Learn++ .NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes // IEEE Transactions on Neural Networks. – 2009. – Vol. 20(1), pp. 152–168.
70. Muhlbaier, M. D., Polikar, R. Multiple classifiers based incremental learning algorithm for learning in nonstationary environments // Proceedings of the Sixth

International Conference on Machine Learning and Cybernetics, ICMLC. – 2007. – Vol. 6. – pp. 3618–3623.

71. Ditzler, G., Polikar, R. An ensemble based incremental learning framework for concept drift and class imbalance // The 2010 International Joint Conference on Neural Networks (IJCNN). – 2010. – pp. 1–8.
72. Ditzler, G., Polikar, R. Incremental Learning of Concept Drift from Streaming Imbalanced Data.IEEE // Transactions on Knowledge and Data Engineering. – 2013. – Vol. 25(10). – pp. 2283–2301.
73. Liao, J., Zhang, J., Ng, W. W. Y. Effects of different base classifiers to Learn++ family algorithms for concept drifting and imbalanced pattern classification problems // Proceedings - International Conference on Machine Learning and Cybernetics. – 2016. – Vol. 1. – pp. 99–104.
74. Iwashita, A. S., Papa, J. P. An Overview on Concept Drift Learning // IEEE Access. – 2019. – Vol. 7. – pp. 1532–1547.
75. Haque, A., Khan, L., Baron, M. SAND: Semi-supervised adaptive novel class detection and classification over data stream // 30th AAAI Conference on Artificial Intelligence, AAAI 2016. – 2016. – pp. 1652–1658.
76. Haque, A., Khan, L., Baron, M., Thuraisingham, B., Aggarwal, C. Efficient handling of concept drift and concept evolution over Stream Data // 2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016. – 2016. – pp. 481–492.
77. Pinagé, F., Santos, E. M. dos, Gama, J. A drift detection method based on dynamic classifier selection. // Data Mining and Knowledge Discovery. – 2020. – Vol. 34(1). – pp. 50–74.
78. Spinosa, E. J., De Carvalho, A. P. D. L. F., Gama, J. OLINDDA: A cluster-based approach for detecting novelty and concept drift in data streams // Proceedings of the ACM Symposium on Applied Computing. – 2007. – pp. 448–452.
79. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G. Learning under Concept Drift: A Review // IEEE Transactions on Knowledge and Data Engineering. – 2019. – Vol. 31(12) . – pp. 2346–2363.

80. Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., Ghédira, K. Discussion and review on evolving data streams and concept drift adapting // Evolving Systems. – 2018. – Vol. 9(1). – pp. 1-23.
81. Hu, H., Kantardzic, M., Sethi, T. S. No Free Lunch Theorem for concept drift detection in streaming data classification: A review // WIREs Data Mining and Knowledge Discovery. – 2020. – Vol. 10(2). – pp. 1-19.
82. Sethi, T. S., Kantardzic, M. Don't pay for validation: Detecting drifts from unlabeled data using Margin Density // Procedia Computer Science. – 2015. – Vol. 53(1) . – pp. 103–112.
83. Sethi, T. S., Kantardzic, M. On the reliable detection of concept drift from streaming unlabeled data // Expert Systems with Applications. – 2017. – Vol. 82. – pp. 77-99.
84. Sethia, T. S., Kantardzic, M. Handling adversarial concept drift in streaming data. // Expert Systems with Applications. – 2018. – Vol. 97. – pp. 18-40.
85. Costa, A. F. J., Albuquerque, R. A. S., Santos, E. M. dos. A Drift Detection Method Based on Active Learning // 2018 International Joint Conference on Neural Networks. – 2018. – pp. 1-8.
86. Zliobaite, I., Bifet, A., Pfahringer, B., Holmes, G. Active learning with drifting streaming data // IEEE Transactions on Neural Networks and Learning Systems. – 2014. – Vol. 25(1). – pp. 27–39.
87. Bashir, S. A., Petrovski, A., Doolan, D. A framework for unsupervised change detection in activity recognition // International Journal of Pervasive Computing and Communications. – 2017. – Vol. 13(2). – pp. 157–175.
88. Liu, A., Lu, J., Liu, F., Zhang, G. Accumulating regional density dissimilarity for concept drift detection in data streams // Pattern Recognition. – 2018. – Vol. 76. – pp. 256–272.
89. Li, B., Wang, Y. jie, Yang, D. sheng, Li, Y. mou, Ma, X. kong. FAAD: an unsupervised fast and accurate anomaly detection method for a multi-dimensional sequence over data stream // Frontiers of Information Technology and Electronic Engineering. – 2019. – Vol. 20(3). – pp. 388–404.

90. Maletzke, A. G., Reis, D. M. dos, Batista, G. E. A. P. A. Quantification in data streams: Initial results // Brazilian Conference on Intelligent Systems (BRACIS). – Uberlandia, Brazil, 2017. – pp. 43–48.
91. Maletzke, A. G., Reis, D. M. dos, Batista, G. E. A. P. A. Combining instance selection and self-training to improve data stream quantification // Journal of the Brazilian Computer Society. – 2018. – Vol. 24(1).
92. Reis, D. Dos, Flach, P., Matwin, S., Batista, G. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test // Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2016. – pp. 1545–1554.
93. Koh, Y. S. CD-TDS: Change detection in transactional data streams for frequent pattern mining // Proceedings of the International Joint Conference on Neural Networks. – 2016. – pp. 1554–1561.
94. Lughofer, E., Weigl, E., Heidl, W., Eitzinger, C., Radauer, T. Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances // Information Sciences. – 2016. – Vol. 355–356. – pp. 127–151.
95. Mustafa, A. M., Ayoade, G., Al-Naami, K., Khan, L., Hamlen, K. W., Thuraisingham, B., Araujo, F. Unsupervised deep embedding for novel class detection over data stream // 2017 IEEE International Conference on Big Data (Big Data). – Boston, MA, USA, 2017. – pp. 1830–1839.
96. de Mello, R. F, Vaz, Y., Grossi, C. H., Bifet, A. On learning guarantees to unsupervised concept drift detection on data streams // Expert Systems with Applications. – 2019. – Vol. 117. – pp. 90–102.
97. Kim, Y., Park, C. H. An efficient concept drift detection method for streaming data under limited labeling // IEICE Transactions on Information and Systems. – 2017. – Vol. E100D(10). – pp. 2537–2546.
98. Yong, B. X., Fathy, Y., Brintrup, A. Bayesian Autoencoders for Drift Detection in Industrial Environments // 2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT. – Roma, Italy, 2020. – pp. 627–631.

99. Jaworski, M., Duda, P., Rutkowski, L. On applying the Restricted Boltzmann Machine to active concept drift detection // 2017 IEEE Symposium Series on Computational Intelligence (SSCI). – Honolulu, HI, USA, 2017. – pp. 1–8.
100. Jaworski, M., Rutkowski, L., Angelov, P. Concept Drift Detection Using Autoencoders in Data Streams Processing // Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). – 2020. – Vol. 12415. – pp. 124–133.
101. Menon, A. G., Gressel, G. Concept Drift Detection in Phishing Using Autoencoders // Communications in Computer and Information Science. – 2021. – Vol. 1366. – pp. 208–220.
102. Шелухин, О. И., Барков, В. В. Экспериментальные исследования и создание базы данных сетевого трафика мобильных устройств под управлением операционной системы Android // Фундаментальные проблемы радиоэлектронного приборостроения: «INTERMATIC-2018». – М.: МИРЭА. – 2018. – Т. 18. – № 4. – С. 1011-1017.
103. Шелухин, О. И., Барков, В. В. Методы сбора сетевого трафика с мобильных устройств под управлением операционной системы Android с целью классификации по типам приложений // Сб. тр. XII Межд. отраслевой науч.-тех. конф. «Технологии информационного общества» (14-15 марта 2018 г., Москва). – М.: МТУСИ. – Т. 2. – С. 20-21.
104. Sheluhin, O. I., Erokhin, S. D., Osin, A. V., Barkov, V. V. Experimental Studies of Network Traffic of Mobile Devices with Android OS / O.I. Sheluhin, S.D. Erokhin, A.V. Osin, V.V. Barkov // Systems of Signals Generating and Processing in the Field of on Board Communications. – 2019.
105. Шелухин, О. И., Ерохин, С. Д., Барков, В. В. Создание базы данных сетевого трафика для автоматизации классификации мобильных приложений под управлением операционной системы Android // Нейрокомпьютеры: разработка, применение. – 2019. – № 1. – С. 40-51.
106. Шелухи, О. И., Барков, В. В., Полковников, М. В. Сравнительный анализ алгоритмов оценки количества и структуры атрибутов в задачах

классификации мобильных приложений // Наукоемкие технологии в космических исследованиях Земли. – 2019. – Т. 11. – № 2. – С. 90–100.

107. Шелухи, О. И., Барков, В. В., Полковников, М. В. Классификация зашифрованного трафика мобильных приложений методом машинного обучения // Вопросы кибербезопасности. – 2018. – № 4(28). – С. 21-28.

108. Барков, В. В., Полковников, М. В. Исследование алгоритмов классификации зашифрованного трафика мобильных устройств по типам приложений методами машинного обучения // Тр. межд. научно-тех. конф. «Телекоммуникационные и вычислительные системы 2018», г. Москва, МТУСИ. – М.: Горячая линия-Телеком. – 2018. – С. 310-312.

109. Sheluhin, O. I., Barkov, V. V. Influence of Background Traffic on the Effectiveness of Mobile Applications Traffic Classification Using Data Mining Techniques // T-Comm. – 2018. – Vol. 12. – no. 10. – pp. 52-55.

110. Шелухин, О. И., Барков, В. В., Маторин, Ф. А. Повышение эффективности классификации противоправных и нежелательных приложений в условиях фонового трафика с помощью автокодировщиков // Вестник Санкт-Петербургского государственного университета технологии и дизайна: Серия 1. Естественные и технические науки. – 2023. – № 3. – С. 90–100.

111. Шелухин, О. И., Барков, В. В., Симонян, А. Г. Обнаружение дрейфа концепта при классификации мобильных приложений с использованием автокодировщиков // Наукоемкие технологии в космических исследованиях Земли. – 2023. – Т. 15. – № 3. – С. 20–29.

112. Шелухин, О. И., Барков, В. В., Секретарёв, С. А. Алгоритмы обнаружения дрейфа концепта при потоковой классификации трафика мобильных приложений // REDS: Телекоммуникационные устройства и системы. – 2020. – № 3. – С. 19-27.

113. Барков, В. В., Секретарёв, С. А. Классификация трафика мобильных устройств по типам приложений методами машинного обучений в режиме онлайн // Межд. форум информатизации (МФИ-2018) 6 Тр. конф.

«Телекоммуникационные и вычислительные системы 2018», г Москва, МТУСИ. – М.: Горячая линия-Телеком. – 2018. – С. 319-321.

114. Sheluhin, O. I., Barkov, V. V., Sekretarev, S. A. The Online Classification of the Mobile Applications Traffic Using Data Mining Techniques. // T-Comm. – 2019. – Vol. 13. – No. 10. – pp. 60-67.

115. Барков, В. В. Классификация трафика нежелательных мобильных приложений методом машинного обучения в потоковом режиме // Сб. научн. трудов II Всерос. науч. школы-семинара «Современные тенденции развития методов и технологии защиты информации». – М., 2022. – С. 167-174.

116. Шелухин, О. И., Барков, В. В. Разработка инфраструктуры для классификации сетевого трафика мобильных приложений с применением алгоритмов машинного обучения // Телекоммуникационные и вычислительные системы - 2017. Тр. межд. научно-тех. конф. – М.: МТУСИ, 2017. – С. 180-181.

117. Барков, В. В. Проектирование и разработка экспертно-аналитической системы "Система анализа трафика" для исследования алгоритмов классификации трафика мобильных устройств под управлением операционной системы Android // Безопасные информационные технологии: Сб. тр. 9-й всерос. науч.-тех. конф. – М.: МГТУ им. Н.Э. Баумана, 2018. – С. 2-12.

118. Шелухин, О. И., Ерохин, С. Д., Барков, В. В. Программный комплекс для онлайн классификации сетевого трафика // Свидетельство о государственной регистрации программы для ЭВМ № 2019615330 от 24 апреля 2019 г.

ПРИЛОЖЕНИЕ А РЕЗУЛЬТАТЫ КЛАССИФИКАЦИИ ПРОТИВОПРАВНЫХ, НЕЖЕЛАТЕЛЬНЫХ И ВРЕДОНОСНЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ АНАЛИЗА СЕТЕВОГО ТРАФИКА ПРИ НАЛИЧИИ И ОТСУТСВИИ ШИФРОВАНИЯ С ПРИМЕНЕНИЕМ АЛГОРИТМОВ KNN И RANDOM FOREST

Без шифрования			С шифрованием		
Приложение	KNN	RF	Приложение	KNN	RF
Precision					
WR	1	1	MI_RU	0,96	0,99
PZ_EPR	0,99	0,99	SP	0,92	0,96
MK	0,99	1	SB	0,93	0,98
GVL	1	0,99	HSN	0,91	0,98
NTV	0,98	0,99	PKB	0,91	0,97
Recall					
WR	0,99	1	MI_RU	0,93	0,97
PZ_EPR	0,99	0,99	SP	0,92	0,98
MK	1	1	SB	0,95	0,99
GVL	0,99	0,99	HSN	0,94	0,99
NTV	0,99	0,99	PKB	0,90	0,96
F1-score					
WR	0,99	1	MI_RU	0,93	0,97
PZ_EPR	0,98	1	SP	0,92	0,98
MK	0,85	1	SB	0,95	0,99
GVL	0,99	1	HSN	0,94	0,99
NTV	0,84	1	PKB	0,90	0,96
Accuracy					
Все	0,99	0,99	Все	0,93	0,98

ПРИЛОЖЕНИЕ Б ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МОДЕЛИ ОБНАРУЖЕНИЯ СМЕНЫ КОНЦЕПТА НА ОСНОВЕ АВТОКОДИРОВЩИКА

Листинг 1 – Исходный код на языке программирования Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from autoencoder.autoencoder import Autoencoder
from enum import Flag, auto
import matplotlib.pyplot as plt
from typing import Any
import os


def calculate_outliers_upper_boundary(data: pd.DataFrame):
    # Quantile Method
    q1 = data.quantile(q=0.25,
                       axis='index',
                       numeric_only=False,
                       interpolation='lower',
                       method='single')
    q3 = data.quantile(q=0.75,
                       axis='index',
                       numeric_only=False,
                       interpolation='lower',
                       method='single')
    iqr = q3 - q1
    return (q3 + 1.5 * iqr).round(4)


class Batch:
    def __init__(self, features_matrix: pd.DataFrame, index: int = -1):
        self._features_matrix = features_matrix
        self._index = index

    def get_data(self):
        return self._features_matrix

    def get_index(self) -> int:
        return self._index

    def get_features_mean(self):
        return self._features_matrix.mean(axis="rows")

    def get_mean(self) -> float:
        return self._features_matrix.mean(axis="columns").mean()

    def get_features_outliers_upper_boundary(self):
        return calculate_outliers_upper_boundary(self._features_matrix)

    def get_outliers_upper_boundary(self):
        return
calculate_outliers_upper_boundary(self._features_matrix.mean(axis="columns"))

    def get_features_instance_threshold_excess_counts(self,
                                                      features_instance_threshold:
pd.DataFrame):
        features_instance_threshold_to_compare: pd.DataFrame = pd.concat(
            [features_instance_threshold.to_frame().T] *
```

```
self._features_matrix.shape[0], axis="rows",
            ignore_index=True).set_index(self._features_matrix.index)
        batch_features_instance_threshold_exceeds: pd.DataFrame = (
                self._features_matrix > features_instance_threshold_to_compare
        )
        return batch_features_instance_threshold_exceeds.sum()

    def get_instance_threshold_excess_count(self, instance_threshold: float):
        batch_features_instance_threshold_exceeds: pd.DataFrame = (
                self._features_matrix.mean(axis="columns") > instance_threshold
        )
        return batch_features_instance_threshold_exceeds.sum()


class ThresholdValidator:
    def __init__(self, name: str, values, thresholds):
        self._name = name
        self._values = values
        self._thresholds = thresholds

    def is_exceeded(self, feature):
        result = self._values[feature] > self._thresholds[feature]
        print(
            f"{self._name} for {feature} is exceeded: {result}"
            f" (value = {self._values[feature]}; threshold =
{self._thresholds[feature]})"
        )
        return result


class Threshold(Flag):
    BATCH_THRESHOLD = auto()
    COUNT_THRESHOLD = auto()


class ThresholdValidators:
    def __init__(self,
                 batch: Batch,
                 batch_threshold_validator: ThresholdValidator,
                 count_threshold_validator: ThresholdValidator):
        self._batch = batch
        self._batch_threshold_validator = batch_threshold_validator
        self._count_threshold_validator = count_threshold_validator

    def get_batch(self):
        return self._batch

    def is_all_exceeded(self, threshold: Threshold, feature):
        print(f"Check thresholds for batch {self._batch.get_index()}")
        result = True
        if Threshold.BATCH_THRESHOLD in threshold:
            result = result and
self._batch_threshold_validator.is_exceeded(feature)
        if Threshold.COUNT_THRESHOLD in threshold:
            result = result and
self._count_threshold_validator.is_exceeded(feature)
        return result


class FeatureDetectInfo:
    def __init__(self,
                 target_value: int,
                 feature,
                 batches_mean_loss: list[float],
```

```
                 batches_loss_instance_threshold_excess_count: list[int],
                 batch_threshold: float,
                 count_threshold: int,
                 warning_threshold: int,
                 detect_threshold: int,
                 warnings: tuple[tuple[ThresholdValidators, ...], ...],
                 detects: tuple[tuple[ThresholdValidators, ...], ...]):
        self._target_value = target_value
        self._feature = feature
        self._batches_mean_loss = batches_mean_loss
        self._batches_loss_instance_threshold_excess_count =
batches_loss_instance_threshold_excess_count
        self._batch_threshold = batch_threshold
        self._count_threshold = count_threshold
        self._warning_threshold = warning_threshold
        self._detect_threshold = detect_threshold
        self._warnings = warnings
        self._detects = detects

    def get_target_value(self):
        return self._target_value

    def get_feature(self):
        return self._feature

    def get_batches_mean_loss(self) -> list[float]:
        return self._batches_mean_loss

    def get_batches_loss_instance_threshold_excess_count(self) -> list[int]:
        return self._batches_loss_instance_threshold_excess_count

    def get_batch_threshold(self) -> float:
        return self._batch_threshold

    def get_count_threshold(self) -> int:
        return self._count_threshold

    def get_warning_threshold(self) -> int:
        return self._warning_threshold

    def get_detect_threshold(self) -> int:
        return self._detect_threshold

    def has_warnings(self) -> bool:
        return len(self._warnings) > 0

    def has_detects(self) -> bool:
        return len(self._detects) > 0

    def get_warnings(self) -> tuple[tuple[ThresholdValidators, ...], ...]:
        return self._warnings

    def get_detects(self) -> tuple[tuple[ThresholdValidators, ...], ...]:
        return self._detects

    def __str__(self):
        return (f"warnings: {self._warnings}"
                f"\ndetects: {self._detects}")


class TargetDetectInfo:
    def __init__(self, target_value: int, features_detect_info:
list[FeatureDetectInfo]):
        self._target_value = target_value
```

```
        self._features_detect_info = features_detect_info

    def get_features_detect_info(self) -> dict[Any, FeatureDetectInfo]:
        return {item.get_feature(): item for item in self._features_detect_info}

    def get_target_value(self):
        return self._target_value

    def has_detects(self):
        for feature_detect_info in self._features_detect_info:
            if feature_detect_info.has_detects():
                return True
        return False

    def has_warnings(self):
        for feature_detect_info in self._features_detect_info:
            if feature_detect_info.has_warnings():
                return True
        return False


class DetectInfo:
    def __init__(self, targets_detect_info: list[TargetDetectInfo]):
        self._targets_detect_info = targets_detect_info

    def get_targets_detect_info(self) -> dict[int, TargetDetectInfo]:
        return {item.get_target_value(): item for item in
self._targets_detect_info}

    def has_detects(self):
        for target_detect_info in self._targets_detect_info:
            if target_detect_info.has_detects():
                return True
        return False

    def has_warnings(self):
        for target_detect_info in self._targets_detect_info:
            if target_detect_info.has_warnings():
                return True
        return False


class Window:
    def __init__(self, feature, warning_threshold: int, detect_threshold: int):
        self._feature = feature
        self._warning_threshold = warning_threshold
        self._detect_threshold = detect_threshold
        self._current_list: list[ThresholdValidators] = []
        self._counter = 0
        self._warning_list: list[tuple[ThresholdValidators, ...]] = []
        self._detect_list: list[tuple[ThresholdValidators, ...]] = []

    def reset(self):
        self._current_list = []
        self._counter = 0
        self._warning_list = []
        self._detect_list = []

    def _finish_handling_current_list(self):
        if self.is_warning():
            self._warning_list.append((*self._current_list,))
        if self.is_detect():
            self._detect_list.append((*self._current_list,))
        self._current_list = []
```

```
        self._counter = 0

    def handle(self, threshold_validators: ThresholdValidators):
        if not threshold_validators.is_all_exceeded(Threshold.BATCH_THRESHOLD,
self._feature):
            self._finish_handling_current_list()
            return
        self._current_list.append(threshold_validators)
        if threshold_validators.is_all_exceeded(Threshold.COUNT_THRESHOLD,
self._feature):
            self._counter += 1

    def detect(self) -> tuple[
        tuple[tuple[ThresholdValidators, ...], ...],
        tuple[tuple[ThresholdValidators, ...], ...]
    ]:
        current_warnings = []
        current_detects = []
        if self.is_warning():
            current_warnings.append((*self._current_list,))
        if self.is_detect():
            current_detects.append((*self._current_list,))
        return (*self._warning_list, *current_warnings), (*self._detect_list,
*current_detects)

    def is_warning(self):
        return len(self._current_list) >= self._warning_threshold and not
self.is_detect()

    def is_detect(self):
        return self._counter >= self._detect_threshold


class FeaturesDriftDetector:
    def __init__(self, target_value: int, batch_size: int, warning_threshold: int,
detect_threshold: int):
        self._target_value = target_value
        self._batch_size = batch_size
        self._warning_threshold = warning_threshold
        self._detect_threshold = detect_threshold
        self._features = None
        self._features_batch_thresholds = None
        self._features_count_thresholds = None
        self._features_instance_threshold = None

    def get_batches_count(self, data: pd.DataFrame) -> int:
        return (
            data.shape[0] // self._batch_size
            if data.shape[0] % self._batch_size == 0
            else data.shape[0] // self._batch_size + 1
        )

    def get_batch(self, data: pd.DataFrame, n: int) -> Batch:
        return Batch(data.iloc[n * self._batch_size: min((n + 1) *
self._batch_size, data.shape[0]), :], n)

    @staticmethod
    def calculate_features_loss_matrix(original_data: pd.DataFrame,
transformed_data: pd.DataFrame) -> pd.DataFrame:
        return (original_data - transformed_data) ** 2

    def fit(self, original_data: pd.DataFrame, transformed_data: pd.DataFrame):
        self._features = list(original_data)
        features_loss_matrix: pd.DataFrame =
```

```
self.calculate_features_loss_matrix(original_data, transformed_data)

        batch_count = self.get_batches_count(features_loss_matrix)
        features_outliers_upper_boundary_for_first_batches = []
        for i in range(0, batch_count):
            batch = self.get_batch(features_loss_matrix, i)
            features_outliers_upper_boundary_for_first_batches.append(
                batch.get_features_outliers_upper_boundary().to_frame().T
            )
        self._features_instance_threshold = pd.concat(
            features_outliers_upper_boundary_for_first_batches, axis='rows',
ignore_index=True).mean(axis='rows')

        batches_features_mean_loss = []
        batches_instance_count_that_exceeds_instance_thresholds = []
        for i in range(0, batch_count):
            batch: Batch = self.get_batch(features_loss_matrix, i)

batches_features_mean_loss.append(batch.get_features_mean().to_frame().T)

            batches_instance_count_that_exceeds_instance_thresholds.append(
                batch.get_features_instance_threshold_excess_counts(
                    self._features_instance_threshold).to_frame().T
            )

        self._features_batch_thresholds = calculate_outliers_upper_boundary(
            pd.concat(batches_features_mean_loss, axis="rows", ignore_index=True)
        )
        self._features_count_thresholds =
pd.concat(batches_instance_count_that_exceeds_instance_thresholds,
                                                    axis="rows",

ignore_index=True).max(axis="rows")

    def detect_at_batch_level(self, original_data: pd.DataFrame, transformed_data:
pd.DataFrame) -> list[
            ThresholdValidators]:
        features_loss_matrix: pd.DataFrame =
self.calculate_features_loss_matrix(original_data, transformed_data)
        # TODO: Move to experiments. Add to DetectInfo necessary data
        cm = 1/2.54
        for feature in features_loss_matrix.columns:
            fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(32 * cm, 20 * cm),
constrained_layout=True)
            fig.set_dpi(600)
            fig.suptitle(f"Признак {feature} приложения {self._target_value}")

            ax1.set_title('Оригинальные данные')
            ax1.plot(original_data[feature], color='blue', marker="o",
linestyle=None)
            ax1.set_ylim(0, 100)
            ax1.set_ylabel('Значение атрибута')
            ax1.set_xlabel('Экземпляры')
            ax1.grid(visible=True, which='major', axis='both')

            ax2.set_title('Восстановленные данные')
            ax2.plot(transformed_data[feature], color='blue', marker="o",
linestyle=None)
            ax2.set_ylim(0, 100)
            ax2.set_ylabel('Значение атрибута')
            ax2.set_xlabel('Экземпляры')
            ax2.grid(visible=True, which='major', axis='both')

            ax3.set_title('Квадраты ошибок восстановления')
```

```
            ax3.plot(features_loss_matrix[feature], color='blue')
            ax3.hlines(y=self._features_instance_threshold[feature],
                       xmin=0,
                       xmax=len(features_loss_matrix),
                       colors='r',
                       linestyles='--',
                       lw=2)
            ax3.set_ylabel('Квадрат ошибки восстановления')
            ax3.set_xlabel('Экземпляры')
            ax3.grid(visible=True, which='major', axis='both')
            if not os.path.exists(f"plots/{self._target_value}"):
                os.makedirs(f"plots/{self._target_value}")
            fig.savefig(f"plots/{self._target_value}/{feature}_dist.png")
        # end of todo
        batch_count = self.get_batches_count(features_loss_matrix)
        batches_threshold_validators = []
        for i in range(0, batch_count):
            batch: Batch = self.get_batch(features_loss_matrix, i)

            batch_features_mean_loss = batch.get_features_mean()
            batch_threshold_validator = ThresholdValidator(
                name="batch threshold",
                values=batch_features_mean_loss,
                thresholds=self._features_batch_thresholds
            )

            features_loss_instance_thresholds_excess_counts = (
                batch.get_features_instance_threshold_excess_counts(
                    self._features_instance_threshold
                )
            )
            count_threshold_validator = ThresholdValidator(
                name="count threshold",
                values=features_loss_instance_thresholds_excess_counts,
                thresholds=self._features_count_thresholds
            )

            batches_threshold_validators.append(ThresholdValidators(
                batch=batch,
                batch_threshold_validator=batch_threshold_validator,
                count_threshold_validator=count_threshold_validator)
            )
        return batches_threshold_validators

    def detect_drift(self, batches_threshold_validators:
list[ThresholdValidators], feature) -> FeatureDetectInfo:
        print("*" * 30)
        print(f"Detect drift for feature: {feature}")
        window = Window(feature, self._warning_threshold, self._detect_threshold)

        batches_mean_loss = []
        batches_loss_instance_threshold_excess_count = []
        for batch_threshold_validators in batches_threshold_validators:
            window.handle(batch_threshold_validators)
            batch = batch_threshold_validators.get_batch()
            batches_mean_loss.append(batch.get_features_mean()[feature])
            batches_loss_instance_threshold_excess_count.append(
                batch.get_features_instance_threshold_excess_counts(
                    self._features_instance_threshold
                )[feature]
            )

        warnings, detects = window.detect()
```

```
        print("*" * 30)
        return FeatureDetectInfo(
            self._target_value,
            feature,
            batches_mean_loss,
            batches_loss_instance_threshold_excess_count,
            self._features_batch_thresholds[feature],
            self._features_count_thresholds[feature],
            self._warning_threshold,
            self._detect_threshold,
            warnings,
            detects)

    def detect(self, original_data: pd.DataFrame, transformed_data: pd.DataFrame)
-> TargetDetectInfo:
        batches_threshold_info = self.detect_at_batch_level(original_data,
transformed_data)
        features_detect_info = []
        for feature in self._features:
            features_detect_info.append(self.detect_drift(batches_threshold_info,
feature))
        return TargetDetectInfo(self._target_value, features_detect_info)


class TargetDriftDetector:
    def __init__(self, target_value: int, epochs: int, batch_size: int,
warning_threshold: int,
                 detect_threshold: int):
        self._target_value = target_value
        self._epochs = epochs
        self._batch_size = batch_size
        self._warning_threshold = warning_threshold
        self._detect_threshold = detect_threshold
        self._autoencoder: Autoencoder | None = None
        self._feature_drift_detector: FeaturesDriftDetector | None = None

    def fit(self, data: pd.DataFrame):
        train_valid_data, threshold_calculation_data = train_test_split(data,
test_size=0.1)

        train_data, valid_data = train_test_split(train_valid_data,
test_size=0.125)

        self._autoencoder = Autoencoder((train_data.shape[1],))
        _ = self._autoencoder.fit(train_data, valid_data, self._epochs)
        transformed_threshold_calculation_data =
self._autoencoder.predict(threshold_calculation_data)

        self._feature_drift_detector = FeaturesDriftDetector(self._target_value,
                                                             self._batch_size,

self._warning_threshold,

self._detect_threshold)
        self._feature_drift_detector.fit(threshold_calculation_data,
transformed_threshold_calculation_data)

    def detect(self, data: pd.DataFrame) -> TargetDetectInfo:
        print(f"\n")
        print("-" * 30)
        print(f"\n\nDetect drift for instances when target feature is
{self._target_value}")
        transformed_data = pd.DataFrame(
            data=self._autoencoder.predict(data),
```

```
            index=data.index,
            columns=data.columns
        )
        result = self._feature_drift_detector.detect(data, transformed_data)
        print("-" * 30)
        return result


class FeatureBasedAeEnsembleDriftDetector:
    def __init__(self, epochs: int = 100, batch_size: int = 32, warning_threshold:
int = 3, detect_threshold: int = 5):
        self._target_values_to_drift_detectors_map: dict[int, TargetDriftDetector]
= {}
        self._epochs = epochs
        self._batch_size = batch_size
        self._warning_threshold = warning_threshold
        self._detect_threshold = detect_threshold
        self._scaler = MinMaxScaler(feature_range=(0, 100))

    def fit(self, features: pd.DataFrame, target: pd.DataFrame):
        self._scaler.fit(features.select_dtypes(include=['number']))
        normalized_features = pd.DataFrame(data=self._scaler.transform(features),
                                           index=features.index,
                                           columns=features.columns)
        for target_value in target.unique():
            group = normalized_features[target == target_value]
            target_drift_detector = TargetDriftDetector(target_value,
self._epochs,
                                                        self._batch_size,
                                                        self._warning_threshold,
self._detect_threshold)
            target_drift_detector.fit(group)
            self._target_values_to_drift_detectors_map[target_value] =
target_drift_detector

    def detect(self, features: pd.DataFrame, target: pd.DataFrame) -> DetectInfo:
        normalized_features = pd.DataFrame(data=self._scaler.transform(features),
                                           index=features.index,
                                           columns=features.columns)
        targets_detect_info: list[TargetDetectInfo] = []

        for target_value in target.unique():
            group = normalized_features[target == target_value]
            targets_detect_info.append(

self._target_values_to_drift_detectors_map[target_value].detect(group)
            )

        return DetectInfo(targets_detect_info)
```

ПРИЛОЖЕНИЕ В СВИДЕТЕЛЬСТВО О ГОСУДАРСТВЕННОЙ РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019615330

Программный комплекс для онлайн классификации сетевого трафика

Правообладатель: ***Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики» (МТУСИ) (RU)***

Авторы: ***Ерохин Сергей Дмитриевич (RU), Барков Вячеслав Валерьевич (RU), Шелухин Олег Иванович (RU)***

Заявка № **2019613947**

Дата поступления **11 апреля 2019 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***24 апреля 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

ПРИЛОЖЕНИЕ Г АКТ ОБ ИСПОЛЬЗОВАНИИ В УЧЕБНОМ ПРОЦЕССЕ ФГБОУ ВО МТУСИ НАУЧНЫХ РЕЗУЛЬТАТОВ ДИССЕРТАЦИОННОЙ РАБОТЫ

«УТВЕРЖДАЮ»
Ректор ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования «Московский технический университет связи и информатики»

_____ С.Д.Ерохин

«__» _____ 2023 г.

Акт об использовании в учебном процессе научных результатов диссертационной работы Баркова В.В. «Классификация противоправных и нежелательных мобильных приложений методами машинного обучения в потоковом режиме»

Комиссия в составе:

- начальника отдела планирования и организации учебного процесса МТУСИ Кузнецовой В.А.;
- декана факультета «Кибернетика и информационная безопасность» Иевлева О.П.;
- заведующего кафедрой «Информационная безопасность» Шелухина О.И.

удостоверяет, что в учебном процессе кафедры «Информационная безопасность» при чтении курса лекций и проведении практических занятий по дисциплине «Искусственный интеллект и машинное обучение в кибербезопасности» для бакалавров направления 10.03.01 «Информационная безопасность» используются результаты диссертационного исследования Баркова В.В., а именно разработанные диссертантом алгоритмы классификации трафика мобильных приложений при наличии фонового трафика с использованием автокодировщиков и обнаружения дрейфа концепции с использованием автокодировщиков, а также разработанная платформа для проведения исследования алгоритмов машинного обучения.

Начальник отдела планирования
и организации учебного процесса МТУСИ Кузнецова В.А.

Декан факультета
«Кибернетика и информационная безопасность»
к.т.н. Иевлев О.П.

Заведующий кафедрой
«Информационная безопасность»
д.т.н., профессор Шелухин О.И.

ПРИЛОЖЕНИЕ Д АКТ ОБ ИСПОЛЬЗОВАНИИ РЕЗУЛЬТАТОВ ДИССЕРТАЦИОННОЙ РАБОТЫ В АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

kaspersky

АО «Лаборатория Касперского»
Россия, Москва, 125212
Ленинградское шоссе, д. 39А, стр. 2

+7 495 797 87 00
www.kaspersky.com
www.securelist.com

Акт об использовании результатов диссертационной работы Баркова Вячеслава Валерьевича «Классификация противоправных и нежелательных мобильных приложений методами машинного обучения в потоковом режиме»

11/10/2023

Комиссия в составе:

председателя комиссии:

Амриллоева А.В.
руководителя отдела разработки продуктов для домашних пользователей

членов комиссии:

Стройкова А.А.
ведущего архитектора программного обеспечения
отдела разработки архитектуры продуктов для Windows

Солодовникова А.Ю.
к.т.н., ведущего архитектора программного обеспечения
отдела разработки архитектуры продуктов для Windows

составила настоящий акт о том, что результаты диссертационной работы Баркова Вячеслава Валерьевича «Классификация противоправных и нежелательных мобильных приложений методами машинного обучения в потоковом режиме» могут быть использованы в производственной деятельности в АО «Лаборатория Касперского» при разработке межсетевых экранов, а именно:

1. алгоритм классификации на основе автокодировщиков в задачах классификации трафика мобильных приложений;
2. алгоритм обнаружения дрейфа концепта на основе автокодировщиков в задачах классификации трафика мобильных приложений;
3. исследовательская платформа для проведения экспериментов, связанных с решением задач классификации, в том числе в задачах классификации трафика мобильных приложений.

Применение разработанных алгоритмов и реализующего их программного обеспечения позволит повысить эффективность классификации трафика мобильных приложений в потоковом режиме при наличии в потоке ранее не встречающихся приложений.

Председатель комиссии: _____ Амриллоев А.В.

Члены комиссии: _____ Стройков А.А.

_____ Солодовников А.Ю.